REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	29.Sep.03	DISSERTATION

4. TITLE AND SUBTITLE
"AN INTERCONNECT-CENTRIC APPROACH FOR ADAPTING VOLTAGE AND FREQUENCY IN HETEROGENEOUS SYSTEM-ON-CHIP"

5. FUNDING NUMBERS

6. AUTHOR(S)
CAPT LAFFELY ANDREW J

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
UNIVERSITY OF MASSACHUSETTS AMHERST

8. PERFORMING ORGANIZATION REPORT NUMBER
CI02-1288

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
THE DEPARTMENT OF THE AIR FORCE
AFIT/CIA, BLDG 125
2950 P STREET
WPAFB OH 45433

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION AVAILABILITY STATEMENT
Unlimited distribution
In Accordance With AFI 35-205/AFIT Sup 1

DISTRIBUTION STATEMENT A
Approved for Public Release
Distribution Unlimited

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

20031015 017

14. SUBJECT TERMS	15. NUMBER OF PAGES
	164
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT

THE VIEWS EXPRESSED IN THIS ARTICLE ARE THOSE OF THE AUTHOR AND DO NOT REFLECT THE OFFICIAL POLICY OR POSITION OF THE UNITED STATES AIR FORCE, DEPARTMENT OF DEFENSE, OR THE U.S. GOVERNMENT

An Interconnect-Centric Approach for Adapting Voltage and Frequency in Heterogeneous System-on-a-Chip

A Dissertation Presented

by

Andrew James Laffely

Submitted to the Graduate School of the
University of Massachusetts Amherst in partial fulfillment
of the requirements for the degree of

Doctor of Philosophy

September 2003

Department of Electrical and Computer Engineering

An Interconnect-Centric Approach for Adapting Voltage and Frequency in Heterogeneous System-on-a-Chip

A Dissertation Presented

by

Andrew James Laffely

Approved as to style and content by:

Wayne P. Burleson, Chair

Russell G. Tessier, Member

Csaba Andras Moritz, Member

Charles C. Weems, Outside Member

Seshu B. Desu, Department Head
Electrical and Computer Engineering

To my wife for her loving support, and my children

Acknowledgments

I thank God for the many blessings of my life, especially for my family and all those who I have been close to over the past three years. The completion of this dissertation is the direct result of labors of many organizations and individuals.

The United States Air Force provided the funding and time away from my duties to complete the PhD. Without this extraordinary program it is doubtful I would have ever come back to school full time. NSF provided funding for the tools and supplies required to run the various experiments.

I would like to express my sincere thanks to my adviser Professor Wayne Burleson for his gentle guidance and encouragement throughout this process. I always left your office feeling energized and excited about what I was doing. In addition to my technical education, you have shown me a leadership and teaching style I can use in my future.

I was fortunate to also work closely with Professor Russell Tessier. From my first day on campus you have been a second adviser for me. By setting high standards for my work you have enhanced this dissertation and made me a more competent engineer.

I would also like to thank my committee members Professor Csaba Andras Moritz, and Professor Charles C. Weems. Your mentorship has been invaluable.

In many ways this dissertation would not be possible without the support of my fellow students. Srini Krishnamoorthy and Jian Liang have been with me since the beginning. Without their friendship and guidance this project would have not happened. So many students have contributed to aSoC: Jeevan Chittamuru, Chris Cowell, Jeongseon Euh, Prashant Jain, David Jasinski, Atul Maheshwari, Aiyappan

Natarajan, Ramaswamy Ramaswamy, Manoj Sinha, Sriram Srinivasan, Srividya Srinivasaraghavan, Subramanian Venkatraman, Ning Weng, and Weifeng Xu.

I would like to also thank the faculty and staff at the University. Especially June Daehler in the office for all her help with paperwork, and John Galloway and Scott Lee Bradley for quick and helpful response to my wacky computer problems. My experience here has been delightful.

Finally, I would like to thank my wife for her support. While I was working on this she has single handedly raised our children and managed our lives. Your selfless devotion to our family is inspirational.

andrew laffely

ABSTRACT

AN INTERCONNECT-CENTRIC APPROACH FOR
ADAPTING VOLTAGE AND FREQUENCY IN
HETEROGENEOUS SYSTEM-ON-A-CHIP

SEPTEMBER 2003

ANDREW JAMES LAFFELY

BSEE, UNIVERSITY OF MAINE, 1992

MSEE, UNIVERSITY OF MAINE, 1994

PH.D, UNIVERSITY OF MASSACHUSETTS AMHERST

Directed by: Professor Wayne P. Burleson

This dissertation proposes a power-aware SoC design methodology, which is characterized by four key elements. First, SoC infrastructure is developed specifically to create modularity in both the physical floorplan, and application. Second, a statically scheduled interconnect approach eases physical design, limits network overhead, and assures predictable interconnect behavior. This interconnect approach is well suited for signal processing applications critical to portable electronics, including video and speech coding, graphics, and cryptography. Third, system modularity is exploited for power savings by allowing the independent development and use of reconfigurable processing cores. *Dynamic parameterization* is proposed as a formalism for run-time reconfiguration of these cores. Finally, interconnect behavior monitoring is used to estimate core utilization and control individual voltage and frequency scaling for each core.

This SoC methodology is applied to create and evaluate power-aware infrastructure for the Adaptive System-on-a-Chip (aSoC). Layout level models are implemented to measure performance and verify architectural assumptions. In aSoC the

global floorplan is enforced with specific regions and process layers allocated for cores and interconnect. As a result, infrastructure overhead can be less than 5% depending on the granularity of the desired IP cores. Interconnect mesh regularity allows for efficient use of resources as well as providing fast and predictable communication links. This structure allows for the routing of global interconnect, global clock, and multiple supply grids together in the top three layers of metal.

The combination of *dynamically parameterized* cores and system wide voltage scaling has the potential to reduce power consumption in future SoC devices by more than 90%.

ABSTRACT

AN INTERCONNECT-CENTRIC APPROACH FOR
ADAPTING VOLTAGE AND FREQUENCY IN
HETEROGENEOUS SYSTEM-ON-A-CHIP

SEPTEMBER 2003

ANDREW JAMES LAFFELY

BSEE, UNIVERSITY OF MAINE, 1992

MSEE, UNIVERSITY OF MAINE, 1994

PH.D, UNIVERSITY OF MASSACHUSETTS AMHERST

Directed by: Professor Wayne P. Burleson

This dissertation proposes a power-aware SoC design methodology, which is characterized by four key elements. First, SoC infrastructure is developed specifically to create modularity in both the physical floorplan, and application. Second, a statically scheduled interconnect approach eases physical design, limits network overhead, and assures predictable interconnect behavior. This interconnect approach is well suited for signal processing applications critical to portable electronics, including video and speech coding, graphics, and cryptography. Third, system modularity is exploited for power savings by allowing the independent development and use of reconfigurable processing cores. *Dynamic parameterization* is proposed as a formalism for run-time reconfiguration of these cores. Finally, interconnect behavior monitoring is used to estimate core utilization and control individual voltage and frequency scaling for each core.

This SoC methodology is applied to create and evaluate power-aware infrastructure for the Adaptive System-on-a-Chip (aSoC). Layout level models are implemented to measure performance and verify architectural assumptions. In aSoC the

global floorplan is enforced with specific regions and process layers allocated for cores and interconnect. As a result, infrastructure overhead can be less than 5% depending on the granularity of the desired IP cores. Interconnect mesh regularity allows for efficient use of resources as well as providing fast and predictable communication links. This structure allows for the routing of global interconnect, global clock, and multiple supply grids together in the top three layers of metal.

The combination of *dynamically parameterized* cores and system wide voltage scaling has the potential to reduce power consumption in future SoC devices by more than 90%.

TABLE OF CONTENTS

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Motivation

Recent industry estimates indicate that silicon devices containing over 1.4 billion transistors will be mass produced by 2012 [1]. This proliferation of resources enables the integration of complex system-on-a-chip (SoC) designs containing a wide range of intellectual property (IP) cores. While, in general, this miniaturization improves performance and power consumption, it also paves the way for increased functional demands. This increase in functionality, like the addition of speech, video, and 3D graphics processing in wireless devices, continues to make power consumption a critical design issue. Power consumption is further complicated by the increased cost of on-chip global interconnect and leakage power in deep sub-micron design.

Fortunately, many circuit and system-level power reduction techniques are emerging. On a circuit level, techniques including current sensing [2, 3] and multi-bit signaling [4] for long interconnects focus on reducing the voltage swing of signals or the relative amount of capacitance being switched. Several recent circuits have effectively reduced leakage power through adaptive body biasing of transistors [5, 6]. On a system level, research has focused on reducing data activity, algorithm complexity, clock frequency, and voltage [7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18]. Possibly the most interesting of these techniques involves run-time reconfiguration to exploit content variations in data [5, 9, 10, 13], or trading-off computation quality for energy conservation [14, 15, 19]. To date, however, little has been done to address controlling power consumption at the scale of systems likely by 2012.

This dissertation proposes a hierarchical approach to power-aware SoC. Due to the additive nature of the components of integrated circuit power consumption, many conservation techniques can be applied together, across all levels of abstraction. At the lowest levels, devices and circuits should, when possible, utilize proper sizing and technology features for low-power. Above this level, the architecture of the SoC provides modularity to the design space. Each core in the system can incorporate some degree of "power-awareness"[15] appropriate for the desired target application or families of applications. In addition to the techniques applied within the cores, the SoC framework can be utilized to control a global power reduction scheme. By monitoring chip-wide communications, this scheme applies voltage and frequency scaling to individual cores, in an attempt to exploit imbalances in core utilization.

1.2 Dynamic Parameterization

A backbone to this approach is *dynamic parameterization* [20], as shown in Figure 1.1. Computational parameters, like operating voltage and frequency, bit width, and filter length, provide a simple yet formal way to characterize incremental changes in algorithms, their implementations, and their performance. These changes are measured through metrics - like power consumption, system throughput, and data quality - which can be used to control parameter reconfiguration. *Dynamic parameterization* first attempts to identify the full set of parameters and metrics, which characterize an algorithm and its implementation. Then it provides a formal methodology to quantify their interactions across the range of environmental conditions, input data and user requirements. This exploration creates a map of the potential implementation trade-offs for a given algorithm. Careful evaluation of this map leads to the development of a specific implementation, including the possible degree of power-awareness. The choice of making parameters dynamic is

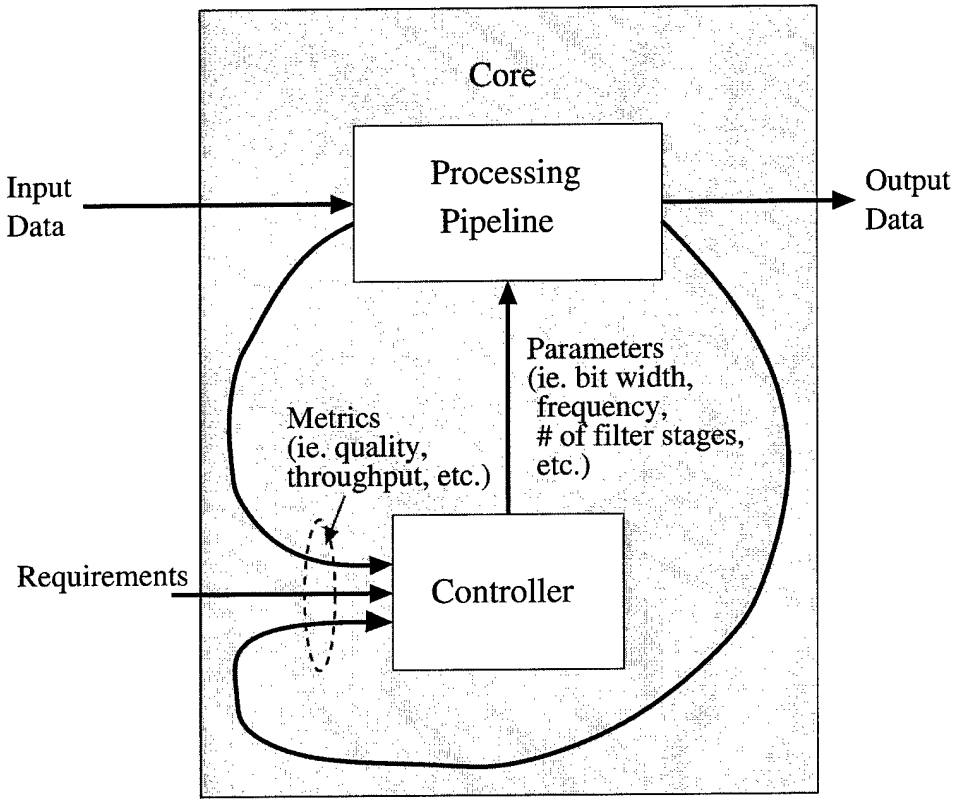


Figure 1.1. Dynamic Parameterized System Model

based on the impact they have on power, measured against the cost of making them run-time flexible. Parameters, whose cost out-weighs the benefits, are fixed at this stage. Each permutation of the now-flexible parameters represents a specific mode of operation. A key component in this approach is the run-time feedback of metrics to control the flexibility of the implementation. Ideally, measurement of these feedback observables should clearly and consistently identify a specific system mode.

1.2.1 Dynamic Parameterization of Cores

In SoC, *dynamic parameterization* can be applied to each core. This dissertation uses a Motion Picture Experts Group (MPEG) [21, 22] motion estimation (ME) core to illustrate the process. The first stage of *dynamic parameterization*, the development of a parameter trade-off map, is taken from P. Jain's Masters thesis

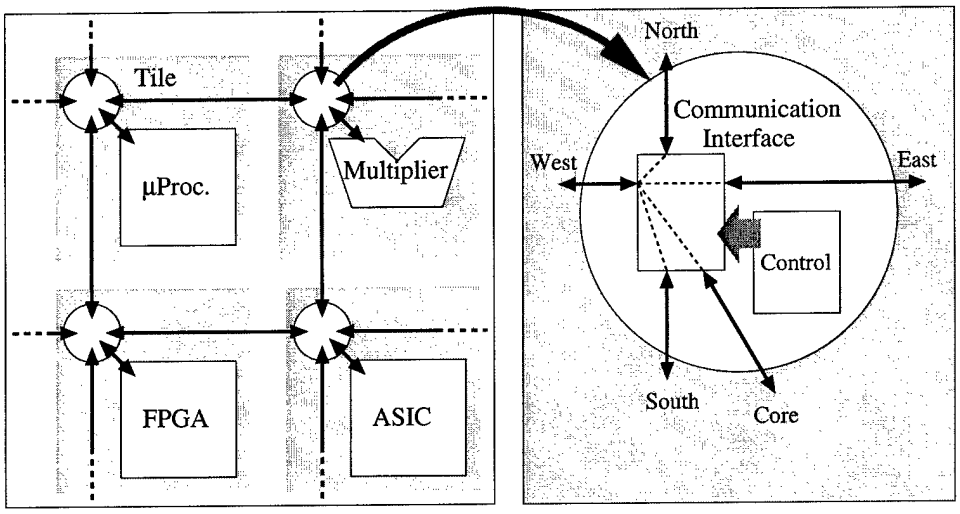


Figure 1.2. Tiled Architecture

[23]. From here, the evaluation of possible metrics clearly identifies a low-cost control mechanism for adaptation. Using this parameterized model reduces core operating power by 60% by removing unneeded computations. As a result, an application-specific core is developed, which can independently control processing time by over an order of magnitude. Other application-specific implementations have shown similar results in power reduction and processing time variations [24, 25]. In fact, it is not uncommon for power-aware cores to implement the control mechanisms locally and use little or no external control [11, 24, 26, 27, 28, 29].

1.2.2 SoC Level Dynamic Parameterization

At the system level, a common approach to SoC integration is the use of tiled architectures to address both scalability and flexibility [30, 31, 32, 33, 34, 35]. As shown in Figure 1.2, each tile represents a computational core and a network interface. This dissertation applies *dynamic parameterization* to the adaptive system-on-a-chip (aSoC) as an example tiled architecture [34]. In aSoC, the core interface supports the use of heterogeneous processing cores occupying one or more tiles. Data is transferred between neighboring tiles in a point-to-point communication pipeline,

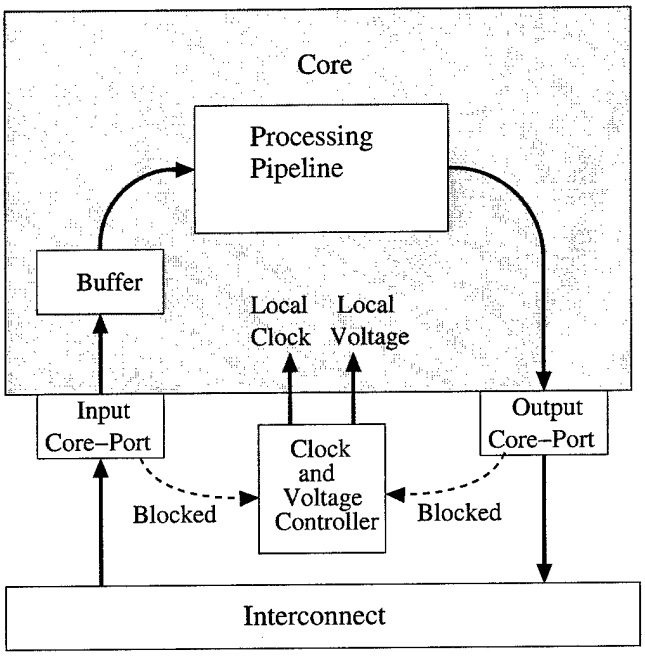


Figure 1.3. SoC Voltage Scaling Approach

thus enabling fast clock rates and time sharing of interconnect resources. Designed as a low-overhead substitute to the on-chip bus, this architecture targets streaming applications by using a statically scheduled mesh of interconnect. Bandwidth in this mesh is allocated to application data streams at compile-time to assure fast and predictable inter-core communication. The result is a communication schedule, which is loaded into each tile interface. Additionally, run-time reconfigurability of both interconnect schedule and resources enables dynamic routing and power management.

In an attempt to simplify *dynamic parameterization* for SoC architectures, the parameters are limited to those associated with the communication and core interface. Interconnect power consumption can be neglected, as it accounts for less then 2% of sytem power [36]. The important parameters are the voltage and frequency supplied to each core. To meet critical-path requirements, independently developed heterogeneous cores already require independent clock and voltage domains.

Additionally, reconfigurable IP cores may require that both the clock and supply voltage be reconfigurable. As a result, much of the overhead for adaptive clock and supply voltage selection already exists in a heterogeneous SoC. Furthermore, the core input and output data ports, as shown in Figure 1.3, present a convenient method for approximating core utilization. The cores in this model are assumed to sequentially process streams of data with little internal buffering. Using this model, the run-time throughput variations of each core create dynamic utilization of the statically scheduled communications bandwidth. For example, a core in a low throughput mode may pull data from its input port slowly and create a blockage in its input data stream. Conversely, if the core switches to a high throughput mode, it could fill up its output data stream and be blocked. The existing aSoC flow control [37] maintains data integrity in the interconnect through the use of valid bits. A controller is added at each aSoC interface, which measures the valid bits to detect blocked streams at the input and output core-ports. This controller interprets core-port blockages to adjust the local operating frequency and voltage of the core. If the input port is repeatedly blocked, the frequency and voltage may be speculatively increased. If the output port is repeatedly blocked, the frequency and voltage may be decreased to save power.

Two other important features are implemented in this approach. First, using the existing interface configuration lines, each local frequency and supply controller can be accessed from the interconnect. This allows the possibility of augmenting the hardware-only speculative scheme with the potential for algorithmic and/or compiler-level support. Second, in addition to measuring blockages at the core interface, the flow control of the interconnect pipeline can be used to find blockages at any point in the interconnect. Measuring data flow at specific points in the interconnect can create additional modularity in the design and help insure the desired throughput of SoC subsystems.

1.3 Contributions

This dissertation will make three main contributions:

- This work proves the physical feasibility of a hardware-speculative approach to power-aware SoC. This dissertation proposes a hierarchical approach to run-time power management in SoC, where subsystems are first made power-aware and then complemented by a global frequency and voltage scaling scheme. This approach leverages the existing hardware demands of heterogeneous SoC to implement a low-overhead voltage and frequency scaling circuit at each core interface. The focus is on the design of SoC hardware to demonstrate the feasibility of hardware speculation. This hardware allows for the speculation of core utilization at run-time by measuring blockages in the interconnect. Additionally, the system supports the use of algorithm and compiler information through an interconnect interface to each frequency and voltage controller. Simple multi-core simulations are run to demonstrate the potential power savings of this hardware speculation. Proving the effectiveness of this approach for many applications is beyond the scope of this document.

- A *dynamically parameterized* MPEG motion estimation system: P. Jain (thesis 2001) [23] created a soft ME core with the flexibility to explore the parameter space. As a result his work produced the trade-off map for array-based ME implementations. This dissertation adds run-time motion vector magnitude speculation to take advantage of the trade-offs in ME. As a result, an autonomous run-time adaptive ME approach is created.

- The first layout-level SoC design methodology and model of the aSoC system: J. Liang and R. Tessier [34] are the primary architects of the aSoC system and developers of the application mapping flow. This work focuses on the

feasibility of the hardware implementation. A layout-level model of the interface is constructed and tested at the functional transistor level using HSPICE. Additionally, standard scaling rules are used to evaluate the aSoC architecture in various technologies. As a result, the aSoC concept is validated at the hardware level and meaningful power and timing data are provided to the architects. In the development of this model, a design methodology for SoC is developed, which specifically addresses hardware design problems.

1.4 Overview

The document proceeds as follows.

- Chapter 2 presents a taxonomy of low-power circuit and system techniques in order to classify and identify those especially pertinent to this work. This chapter attempts to identify and define the terminology used throughout the document. Additionally, voltage scaling, as the main approach proposed, is investigated in more detail to justify the implementation approach of this dissertation. A comparison of two voltage scaling systems, supported by HSPICE simulations, is provided.

- Chapter 3 contains a detailed presentation of *dynamic parameterization.* ME is used to illustrate the process.

- Chapter 4 presents an overview of the SoC architectures. This background section compares the various SoC approaches to motivate the choice of aSoC for use in this study. The architectural details of aSoC are presented as a foundation for subsequent chapters.

- Chapter 5 presents the parameterization of the aSoC interconnect and the resulting hardware-speculative approach to voltage and frequency scaling.

- Chapter 6 shows hardware level results for the layout model of the aSoC interface system. Cadence tools and HSPICE are used to demonstrate realistic system performance.

- Chapter 7 describes the application example, MPEG, which will show the functionality of the system-wide voltage and frequency scaling approach. A cycle-based simulator augmented with power information can be used in this evaluation.

- Chapter 8 presents conclusions and future work.

Chapter 2

Power Reduction

This chapter attempts to identify and define the power-related terminology used throughout the document. It presents a taxonomy of low-power circuit and system design techniques in order to classify and identify those pertinent to this work. Additionally, voltage scaling is investigated in detail to justify the implementation approach of this dissertation.

2.1 Classification of Power Measures

There are many metrics used to represent and quantify aspects of on-chip power consumption. In a broad sense, these can be broken up into three categories: instantaneous, average, and power-delay products. In the first category, metrics involving the maximum instantaneous power or current are often used when designers are concerned about system reliability and lifetime. Large current spikes on the supply rails can cause local values of the supply voltage (v_{dd}) to dip. This dip reduces noise margins and potentially leads to soft errors. In addition, large current spikes cause heating problems and electromigration, which reduce system lifetime. The second category helps quantify the size or lifetime of a portable device's battery. Arguably, both total energy and average power provide this measure. Finally, the third category uses energy-delay and power-delay products to couple system power to a performance metric. These powerful measures give an idea of how much work can be done for a given amount of energy.

Average power and total energy are primarily used in this dissertation, as the system developed strives to improve battery lifetime without impacting system

throughput. Cases where throughput is not maintained will be clearly noted. Power is used when discussing system examples where the clock frequency is fixed. Energy is used when investigating discrete events.

2.2 Taxonomy of Low Power Techniques

Integrated circuit power consumption has been a critical concern over the past decade. As a result, power reduction has been an intensely researched area, and has produced a multitude of reduction techniques [7, 8, 38, 39, 40, 41]. The taxonomy of Figure 2.1 is intended to clearly identify power reduction techniques applicable to this research.

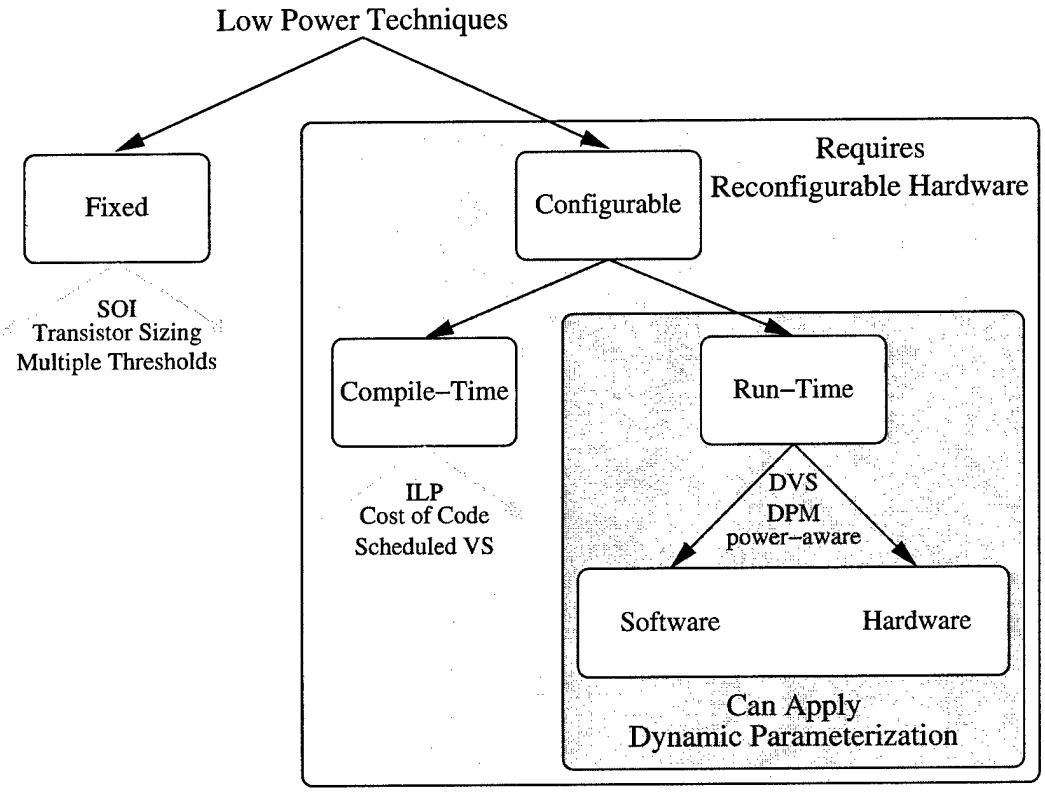


Figure 2.1. Classification of Low Power Techniques

At the highest level in the taxonomy, power reduction techniques belong to one of two distinct classes: *fixed* or *configurable*. The fixed techniques include all those methods, which are designed into the layout or physics of a device and cannot be changed after fabrication. Many of these techniques, including transistor sizing and multiple threshold technologies, have been studied since the early 1990s [7, 8, 38, 39, 40, 41]. More recent fixed techniques include various interconnect signaling techniques [2, 4, 42, 43, 44] and silicon on insulator (SOI) technology [8, 45]. At a system level, low power systems can be developed by evaluating power metrics early in the design process [46]. Computer-aided design (CAD) tools for application-specific integrated circuit (ASIC) development can be modified to reduce system capacitance by enforcing locality [47]. And, if applications require only moderate clock frequencies, subthreshold supply voltage can be applied to CMOS circuits [48] to achieve low power. Due to the additive nature of power consumption, circuit and system techniques can be and should be used together [41].

The other class, configurable techniques, includes all those methods that can be controlled after fabrication. Although the fixed techniques are critical to the development of low-power very large scale integrated (VLSI) systems, this dissertation focuses on the configurable class. This class can further be broken down into two subclasses: *dynamic* and *static*. To avoid confusion with the terms dynamic and static power, this dissertation will refer to these two subclasses as *run-time* and *compile-time* reconfigurability. Compile-time reconfigurable systems fix power settings or schedules of settings before run-time. This may involve selecting power conservative algorithm parameters [49, 50, 51, 52], optimizing system binaries with special compiler techniques that evaluate the instruction level parallelism [53], or evaluating the power cost of code [54, 55, 56, 57]. Several authors have suggested using compile-time information to schedule run-time power reduction techniques, like voltage scaling [9, 56].

Run-time reconfiguration creates power-aware systems by reducing computation, frequency and/or voltage based on changes in data or operating environment. In fact, there are several names for these types of systems. One of the most specific of these approaches, dynamic voltage scaling (DVS), attempts to reduce power by dynamically reducing the supply voltage of the system when possible or necessary [17, 58, 59, 60]. "Dynamic power management" (DPM), from L. Benini and G. De Micheli [9, 10, 11, 12, 13, 26, 59], simplifies voltage scaling to the concept of supply enabling. In this approach idle subsystems are switched off until they are needed. In a more general approach, the widely used term power-aware [14, 15, 19] applies to systems, which attempt to dynamically control one or more of the parameters in the standard VLSI power equation. This could be voltage scaling, but it also includes techniques to reduce clock frequency, switching activity or the capacitance being switched [9, 19]. Even more general is the concept of "approximate signal processing", where the quality of the computation can be traded for some aspect of performance [61]. This technique is not limited to trading quality for power but is an attractive approach to creating power-aware systems [23, 29].

At the heart of these approaches is the use of reconfigurable processing. Using reconfigurable processing, aspects of the system environment or data can be exploited to create performance and/or power trade-offs at run-time [62]. A key issue is the cost of making systems reconfigurable in terms of area, power, and throughput. First, reconfigurable components must not significantly increase circuit area and therefore the manurfacturing cost. Secondly, reconfiguration must not increase power. Both the power cost per configuration and the frequency of reconfiguration must be consisdered. Finally, as systems typically need to stop processing while being reconfigured, the time per configuration and their frequency will negatively impact system throughput. J. Rabaey's group demonstrates the use of reconfigurable components in a low-power digital signal processor (DSP) [63].

The approach in this dissertation, *dynamic parameterization* [20, 24, 28], formalizes the development and control of power-aware systems. Although all power-aware development results in parameterized systems, dynamic parameterization provides a formal, top-down perspective to this problem. Unlike the other approaches, dynamic parameterization presents a detailed methodology for finding reconfigurable parameters and run-time control metrics. As will be seen in Chapter 3, dynamic parameterization can be applied to make arbitrary systems power-aware.

Finally, the class of run-time reconfigurable approaches can be further classified by the use of *software* or *hardware*. In reality this breakdown represents a spectrum of possibilities. At one end of this spectrum, the software-only approaches attempt to creatively utilize the flexibility in the existing hardware to perform run-time reconfiguration. For example, this may involve modifying data access patterns [10]. The software-only approach is slightly blurred by the fact that some of the existing features in modern architectures are designed for run-time reconfigurability. A step removed from the software-only approach is a software-dominated approach. In these systems, small and virtually transparent hardware modifications are proposed and supported with a host of software algorithms [53, 64, 65]. Fairly obviously, the software approaches are dominant in microprocessor, digital signal processor (DSP) and reduced instruction set computer (RISC) systems. At the other end of the spectrum are the ASIC systems. Here, new architectures are proposed for specific algorithms. Often these systems require little or no software support [23, 29, 66, 67]. SoC represents a middle ground in this spectrum. As SoCs may contain processors, custom ASICs, and FPGA sub-systems, it may be possible to use a wide variety of approaches in one device. In fact, with the interdependence of these devices it is not unrealistic for one component to control the reconfiguration of another. This may mean software control of custom components or vice versa.

2.3 Components of Power Consumption in Integrated Circuits

In this section, the standard VLSI power equation, Equation 2.1, is examined with an attempt to justify voltage scaling as the primary method of power reduction in SoC. The material presented here summarizes the discussion in "Low Power Digital CMOS Design" [7]. Circuit simulations using Berkeley Predictive Models [68] enhance the discussion to cover future CMOS technologies down to 70nm.

$$P_{ave} = P_{switch} + P_{shortcircuit} + P_{leakage} + P_{static} \tag{2.1}$$

A simple ring oscillator, shown in Figure 2.2, is used to understand both the voltage scaling properties of the switch and leakage components over several technologies. It has been shown [27, 69] that the ring oscillator accurately characterizes the voltage-delay characteristics of a wide range of circuit types. Several special circuits, shown in Figures 2.7, 2.14, 2.17 and 2.20, are used in the discussion of short circuit and static power. These will be discussed in detail as needed.

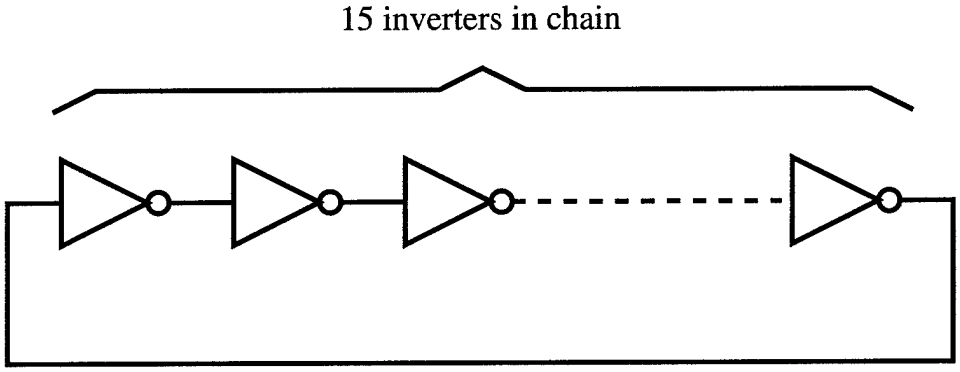


Figure 2.2. Ring Oscillator

2.3.1 Switch Power

Historically, switching power, as shown in Equation 2.2, has dominated integrated circuit power consumption.

$$P_{switch} = \alpha \cdot C \cdot V_{dd}^2 \cdot f \tag{2.2}$$

where,

- α represents the switching activity. It is a percentage of the system capacitance, which is being switched for each clock cycle. As nodes may switch multiple times per clock cycle, α can be greater than one. Typically a value less then 0.5 is assumed [7, 69] for standard CMOS.

- C is the total capacitance of the system. This value tends to increase with technology scaling as circuit die size increases.

- f is the operating frequency of the device. Reducing operating frequency alone implies that less work is being accomplished. Reducing the clock alone eliminates clock tree switching, which, for synchronous systems, may be large [70]. Clock reduction may also reduce internal state changes and the completion of unneeded calculations.

- V_{dd}^2 represents the product of two distinct quantities. First, V_{dd} is the potential difference across the power rails of the device. Second, V_{swing}, when coupled with the capacitance, describes how much charge is moving between the rails. For CMOS, most nodes are assumed to have full swing making $V_{swing} = V_{dd}$.

The squared-dependence of power on supply voltage makes voltage scaling an attractive power reduction technique. Unfortunately the system delay, t_p, is also dependent on supply voltage, V_{dd}, as shown in Equation 2.3.

$$t_p \propto \frac{V_{dd}}{|I_{ave}|} \tag{2.3}$$

Traditionally $|I_{ave}|$ is assumed to be proportional to $(V_{dd} - |V_t|)^n$, where n is 2 for long channel transistors. This makes t_p proportional to $1/V_{dd}$, and results in a linear improvement in power-delay product with voltage supply scaling. In deep sub-micron technologies the relationship for t_p is significantly more complicated. First, n , a factor to account for velocity saturation, is reduced from 2 to nearly 1 as the technology scales to 70nm. Second, the $1/V_{dd}$ dependence assumes V_{dd}, is much greater than the transistor threshold voltage, $|V_t|$. In smaller technologies $|V_t|$ plays a more critical role in delay. Table 2.1 shows the relative increase of $|V_t|$ with respect to V_{dd}. Third, the original development assumes that the average current during switching is equal to the transistor saturation current. In fact, in deep sub-micron technologies the average current has three components: $|I_{ave}| = \alpha I_{sat} + \beta I_{lin} + \gamma I_{leak}$, where $\alpha + \beta + \gamma = 1$. The role of the saturation current, I_{sat}, is reduced as voltage scales down and the drain-to-source voltage, V_{DS}, is less than $|V_t|$ during more of the transition. As a result, the role of the linear current I_{lin} is increased at lower supply voltages. Finally, the leakage current, I_{leak}, component increases in importance as ultra low voltage scaling, as shown in Figure 2.3, creates periods of time during transitions when neither the pull-up or pull-down networks are conducting.

Technology	V_{t_n} NMOS	V_{t_p} PMOS	V_{dd}^*	V_{t_ave}/V_{dd}
$180nm$	0.399	0.42	1.8	.228
$130nm$	0.335	0.35	1.3	.263
$100nm$	0.261	0.303	1.0	.282
$70nm$	0.19	0.21	0.7	.286
* V_{dd} values chosen base on straight technology scaling.				

Table 2.1. Values for V_t for Nominal BPTM[68]

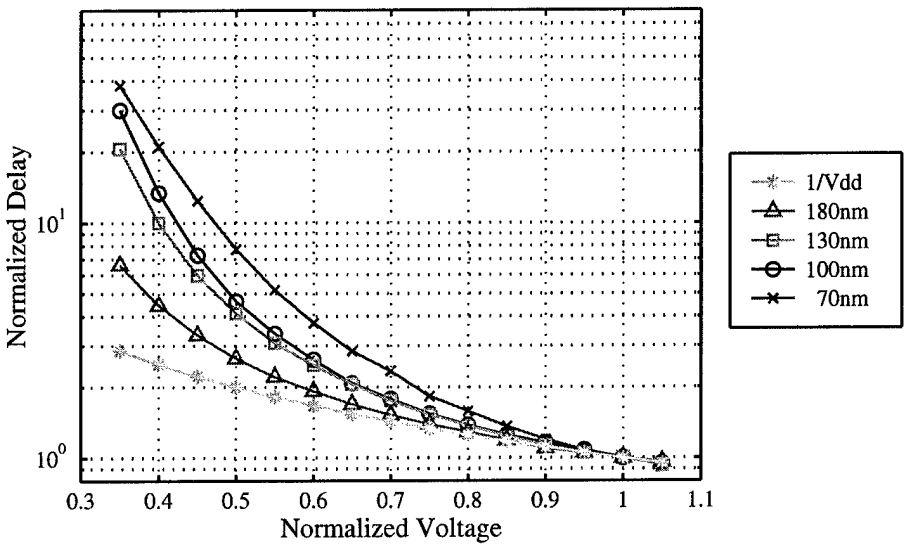


Figure 2.3. Normalized 15-Stage Ring Oscillator Delay as a Function of Voltage for Sub-micron Technologies

Figure 2.3 shows the cost of reducing voltage in terms of delay. V_{dd} is normalized by the values shown in Table 2.1. To emphasize the relative increase in cost, the traditional, $1/V_{dd}$ approximation is also shown. Measured delays from HSPICE simulations of the 15-stage ring oscillator are simply labeled with the technology. All delays are normalized by the value found using the voltage, V_{dd}, from Table 2.1.

Figure 2.4 shows the simulated energy per cycle for the 15-stage ring oscillator as voltage scales. To attain a fair comparison with the theoretical equation, the simulated leakage energy is removed from the total energy. As a result of the fast input rise times and standard CMOS implementation, short circuit and static power are negligible for this circuit. For all technologies, the energy calculated by HSPICE simulations using BPTM [68] match closely to the theoretical V_{dd}^2 predicted in Equation 2.2.

Combining the data from Figures 2.3 and 2.4 shows the relative cost for energy reduction using voltage scaling. Figure 2.5 shows the energy-delay product for the ring oscillator. For all technologies the energy-delay product decreases initially as

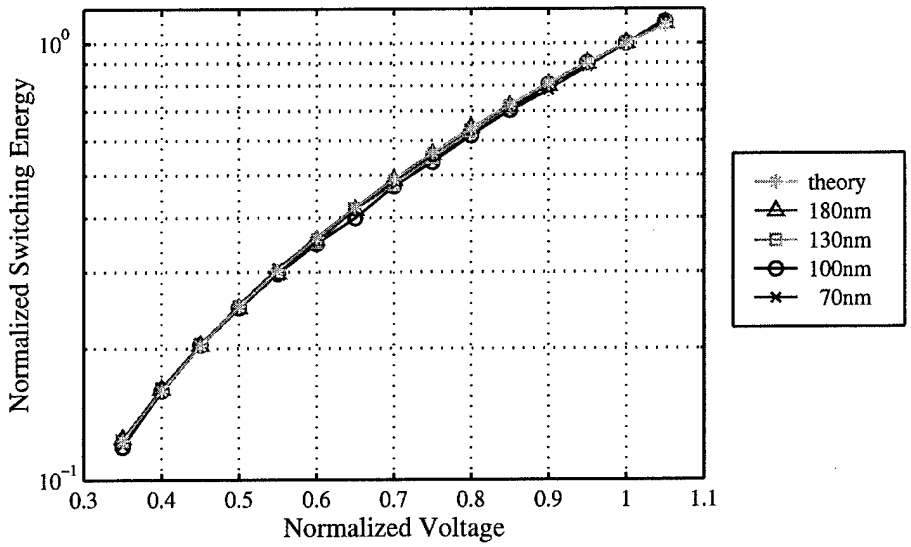


Figure 2.4. Normalized 15-Stage Ring Oscillator Switching Energy as a Function of Voltage for Sub-micron Technologies

the energy decreases more quickly than the increase in delay. As the supply voltage, V_{dd}, approaches the threshold voltage, V_t, the delay increases sharply, causing the energy-delay product to increase as well.

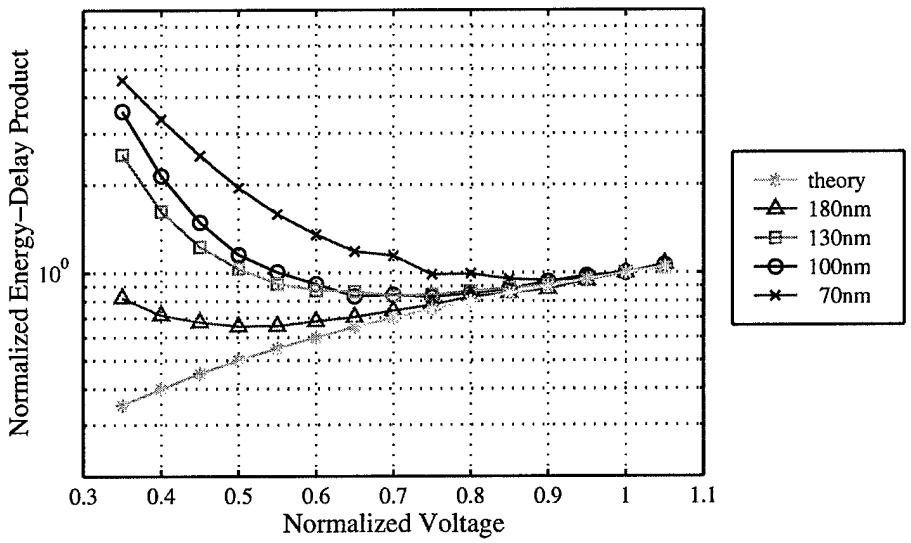


Figure 2.5. Normalized 15-Stage Ring Oscillator Switching Energy-Delay Product as a Function of Voltage for Sub-micron Technologies

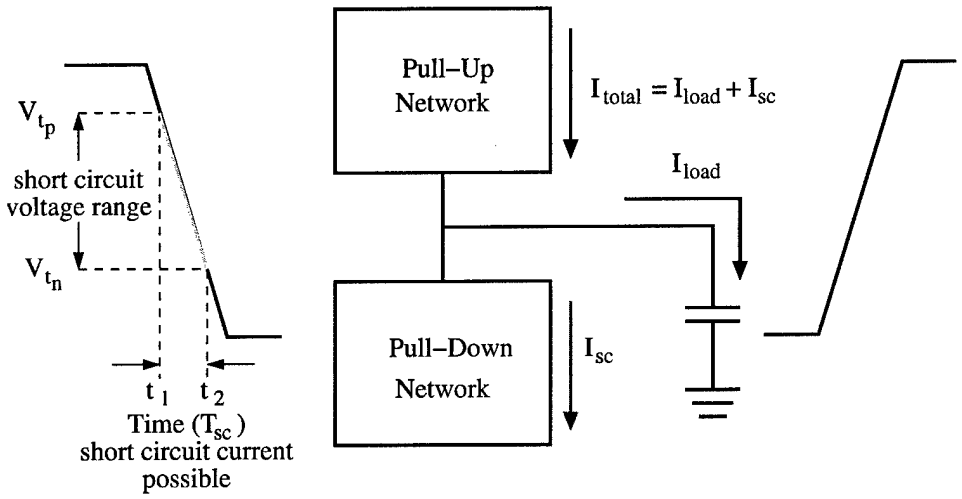


Figure 2.6. Source of Short Circuit Power

2.3.2 Short Circuit Power

When switching, each gate may experience a period of time where both the pull-up and pull-down networks are conducting, as shown in Figure 2.6. From the figure, two important implications can be seen. First, the total short circuit current is dependent on the rise (or fall) time of the input signal. The slower the input transition, the longer the time that both networks are conducting. Second, the short circuit current is eliminated when $V_{dd} < V_{t_n} + |V_{t_p}|$, when the devices can never conduct simultaneously. In "Low Power Digital CMOS Design" [7], A. Chandrakasan bounds the problem by assuming that all current in the time T_{sc} contributes to short circuit power consumption, as shown in Equation 2.4. This approach also assumes the devices are in saturation for the entire period of T_{sc} and therefore is most accurate when the input transition is much slower than the output. The equation presented here has been modified to depict circuits with short channel effects. This is done by lowering the exponent from 3 to 2. To judge voltage scaling based on this equation is somewhat misleading, as supply reduction also changes the rise or fall times of the input and output signals.

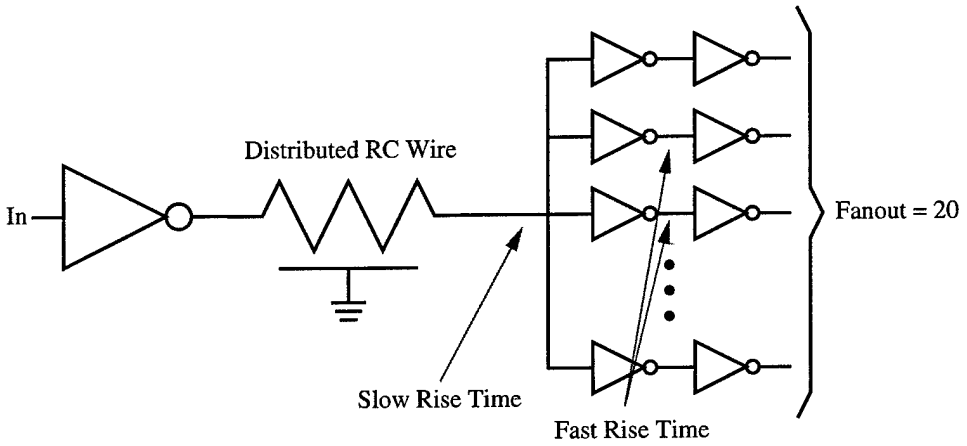


Figure 2.7. Example Circuit with Short Circuit Power Consumption

$$P_{ave} \propto T_{sc} \cdot (V_{dd} - 2V_t)^2 \tag{2.4}$$

Short circuit power is not testable in the ring oscillator, as the input and output transition times are always comparable. Therefore, to test for short circuit power, the interconnect system shown in Figure 2.7 is modeled in HSPICE. In this model, the distributed RC line is approximately $33k\lambda$ for each technology and is modeled as a 5π structure. The interconnect driver is undersized to further slow the transition on the interconnect. Although the delay on the interconnect has been exaggerated to enhance short circuit current measurement, this type of structure is common in bus-based architectures and FPGAs interconnects [71, 72, 73].

Figure 2.8 shows the relative time during which the short circuit current occurs. Note that although the circuit delay increases as voltage reduces, the voltage reduction quickly closes the voltage gap where short circuit conditions can exist. Figure 2.9 shows the normalized energy for 70, 100, 130 and $180nm$ technologies and Equation 2.4 using measured rise times. In general, short circuit power is quickly eliminated as voltage scales down. The slope of decrease in the theoretical predictions is greater than the simulated results as a result of using the saturation current approximation in Equation 2.4.

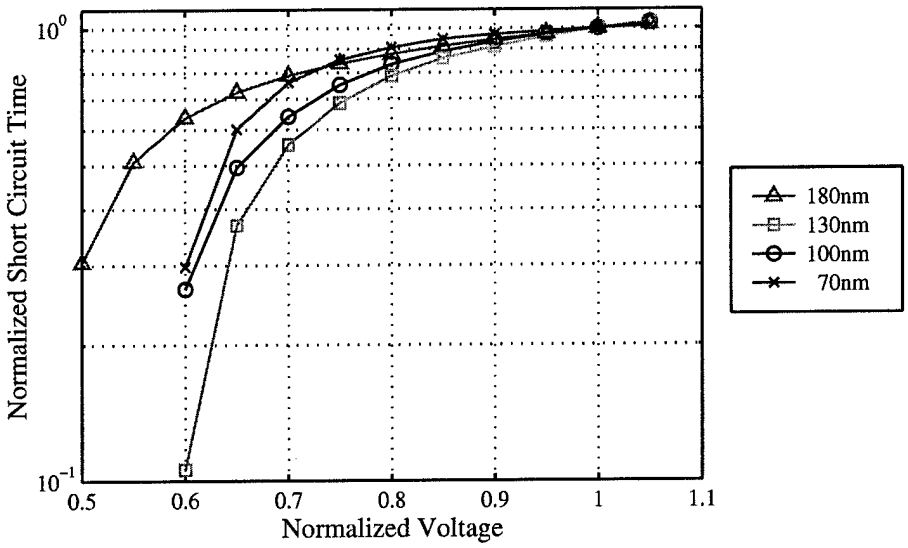


Figure 2.8. Normalized Short Circuit Time as a Function of Voltage for Sub-micron Technologies

The cost of voltage scaling for this circuit in terms of delay is shown in Figures 2.10 and 2.11. For these figures the output is fed back to the input to create an oscillator. The normalized delay shown in Figure 2.10 is based on the oscillation period. The delay increase for this interconnect circuit is comparable to those found

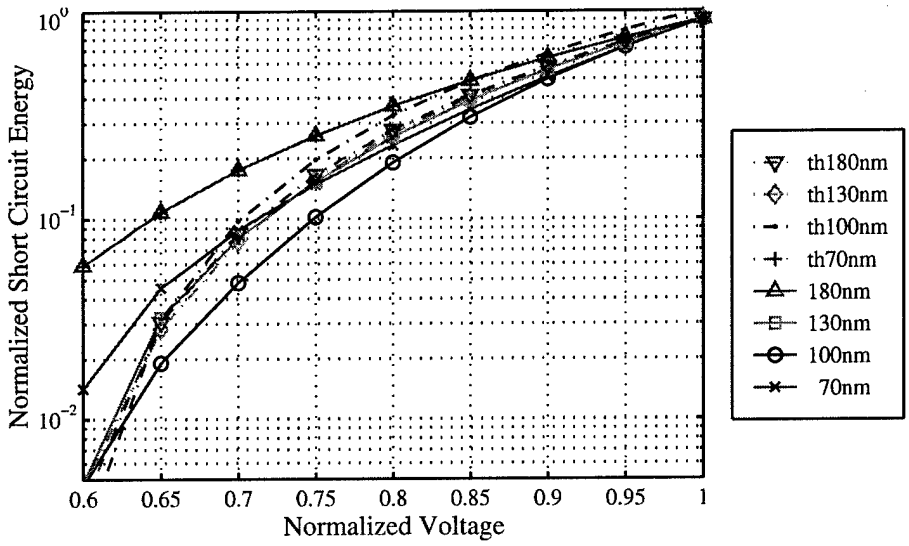


Figure 2.9. Normalized Short Circuit Power as a Function of Voltage for Sub-micron Technologies

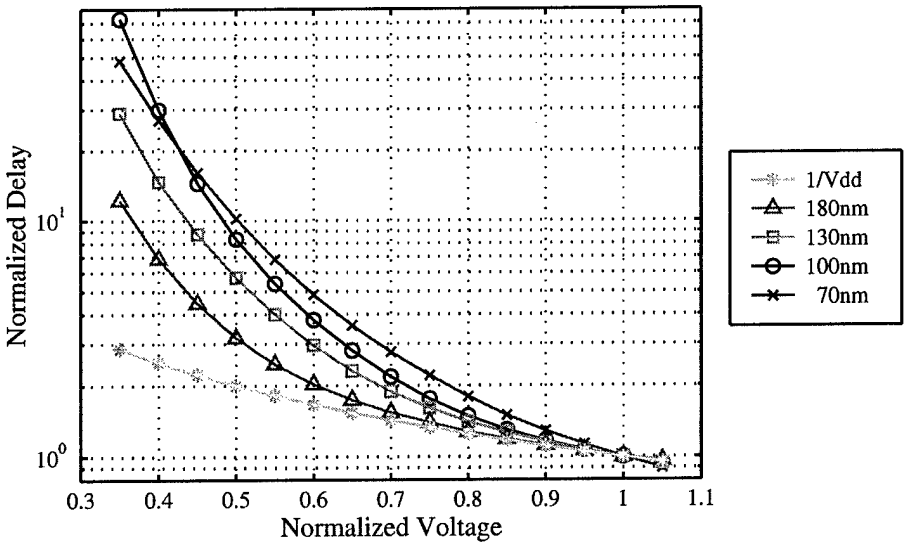


Figure 2.10. Normalized Short Circuit System Delay as a Function of Voltage for Sub-micron Technologies

in the ring oscillator. The energy-delay product, shown in Figure 2.11, predicts an advantage in voltage scaling. However the circuit has been specifically designed to have short circuit power dissipation. In reality, switching power tends to dominate the total power, and the energy-delay closer to that shown in Figure 2.5.

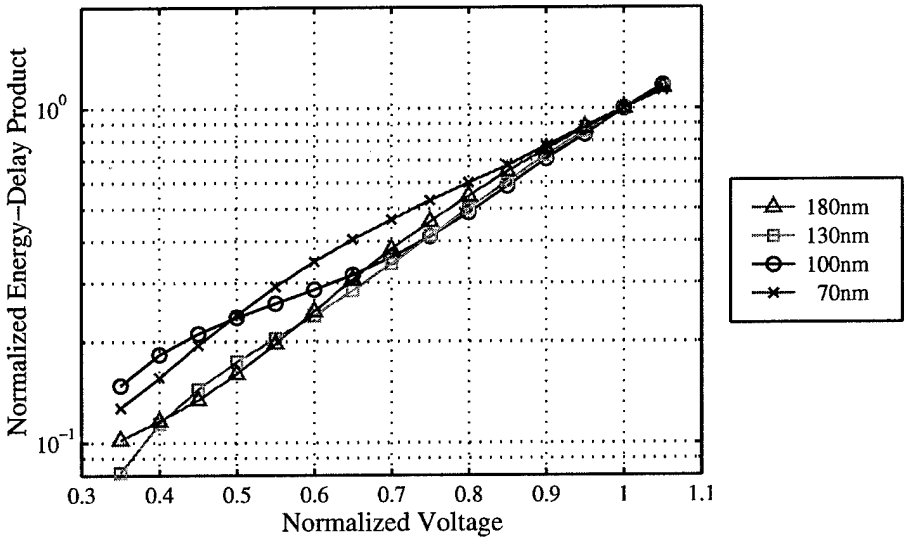


Figure 2.11. Normalized Short Circuit Energy-Delay Product as a Function of Voltage for Sub-micron Technologies

2.3.3 Leakage Power

While there are many types of leakage [74], subthreshold leakage described in Equations 2.5 and 2.6 tends to dominate [74]. The condition is called drain-induced barrier lowering (DIBL), and occurs when the depletion region extends far enough in the channel to interact with the source. This lowers the potential barrier of the source, causing current flow from drain to source. These equations predict that leakage power is proportional to $V_{dd} \times e^{\delta V_{dd}}$, where δ is determined by the process.

$$I_{leak} = Ae^{\frac{1}{nV_T}(V_{GS}-V_{t0}-\gamma' V_S+\eta V_{DS})} \times (1 - e^{V_{DS}/V_T}) \simeq Ae^{\frac{-V_{t0}}{nV_T}} \times e^{\frac{\eta V_{DS}}{nV_T}} \tag{2.5}$$

where A can be given as in [74],

$$A = \mu_0 C_{ox} \frac{W}{L_{eff}} V_T^2 e^{1.8} e^{\frac{-\Delta V_t}{nVT}} \tag{2.6}$$

with V_T (thermal voltage) $= 0.026V$, $V_{GS} = 0V$, and $V_{DS} = V_{dd}$.

Figure 2.12 shows the normalized leakage power for sub-micron technologies. Also shown on the graph are the results of the theoretical equation. For this example, $\delta\times$ the normalized supply voltage is 2. Figure 2.13 shows the ratio of leakage power to the total power for the ring oscillator example. The dynamic power has been scaled by 0.5 to make it more representative of typical CMOS logic. The ratio increases with voltage scaling, as the slope of switching power reduction is greater than that of leakage power.

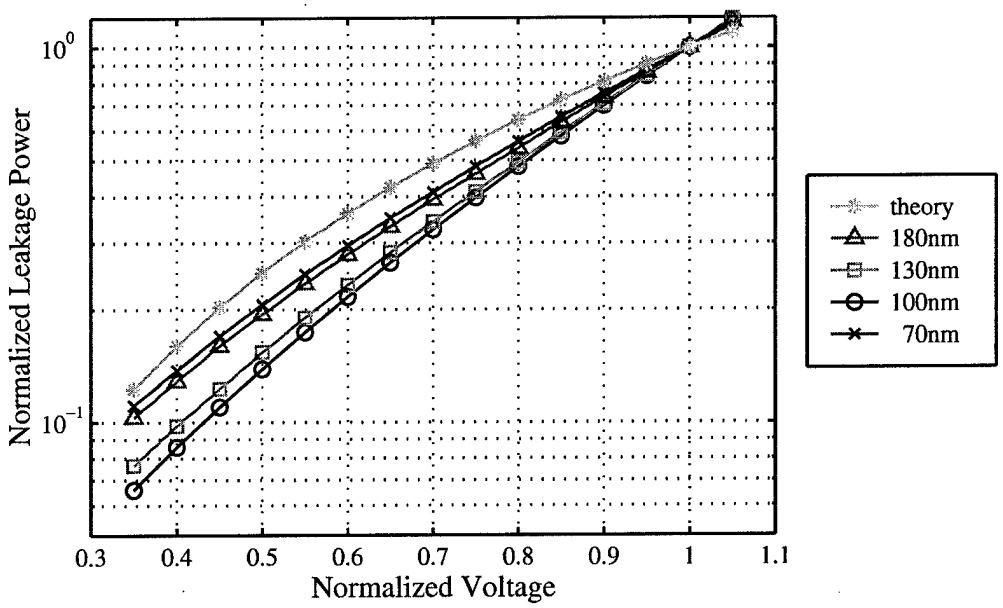


Figure 2.12. Normalized Leakage Power as a Function of Voltage for Sub-micron Technologies

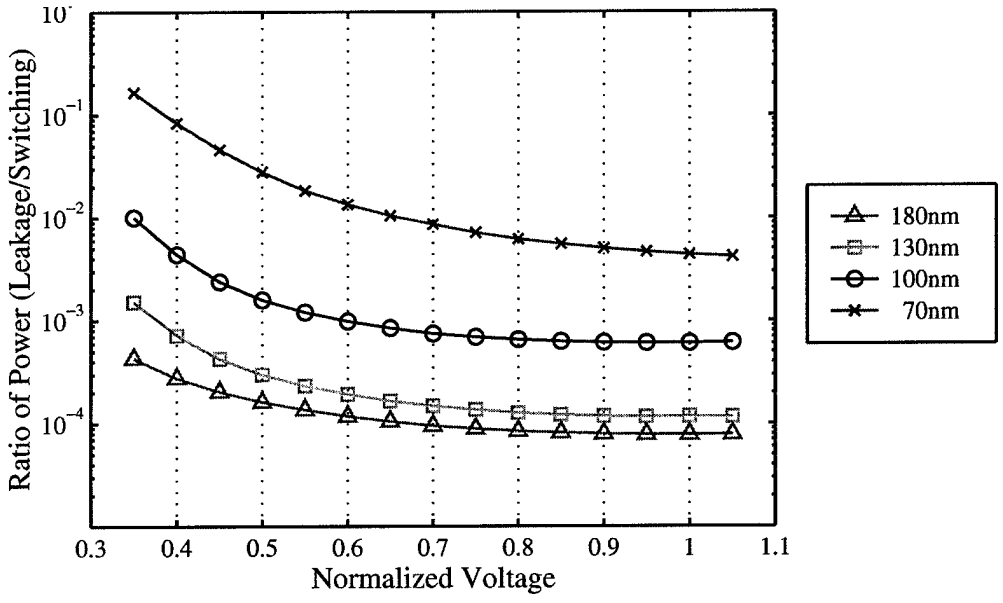


Figure 2.13. Normalized Ratios of Leakage and Switching Power for Sub-micron Technologies

2.3.4 Static Power

Static power dissipation occurs anytime the inputs to CMOS transistors are set to values other then the supply rails. This situation can occur when using pass-transistor logic, shown in Figure 2.14, pre-equalized differential voltage sensing, shown in Figure 2.17, or in more exotic current sensing techniques, like the single-ended current-sense amplifier shown in Figure 2.20 [2]. While pass-transistor logic is widely used, use of analog-like circuits is becoming more popular for the difficult problems of interconnect [3, 75].

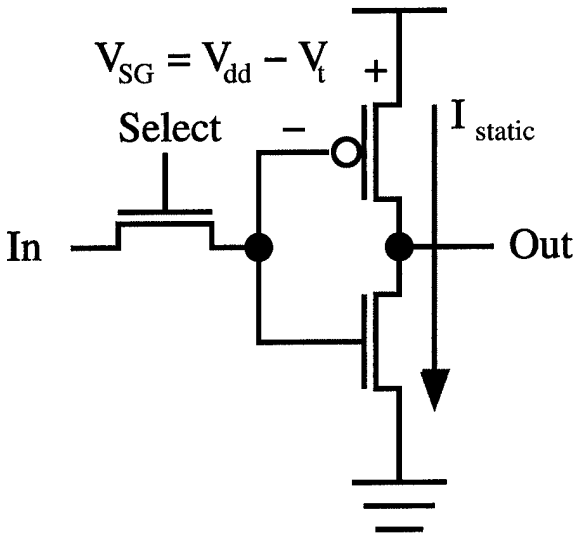


Figure 2.14. Pass Transistor Circuit with Static Power Dissipation

For the pass transistor circuit shown in Figure 2.14, static power dissipation occurs when the input signal is high. In this case the NMOS device is conducting and the output drops to near 0V. Unfortunately the pass gate inflicts a V_t drop on the input signal, causing V_{GS} of the PMOS device to be less than $0V$. For this situation it is difficult to find a theoretical bound for the problem, as the device is at the boundary between cut-off and saturation. The power should be the combination of DIBL leakage and saturation current flow. Figure 2.15 shows the effects of voltage

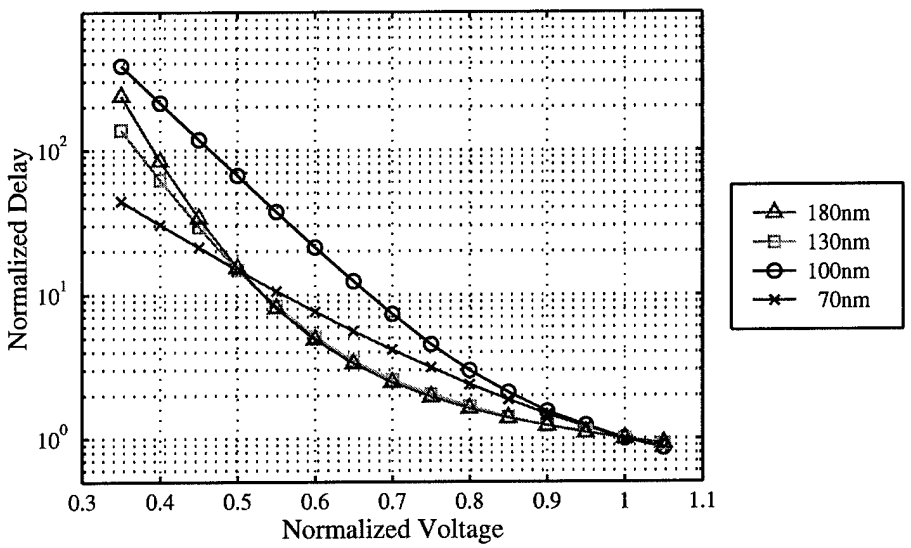


Figure 2.15. Delay of Pass-Gate Logic

scaling on the delay of the pass transistor circuit. In general, the delay induced by voltage scaling is worse in this case than in the standard CMOS system. At the same time, the benefit of voltage scaling on power is much more pronounced, as shown in Figure 2.16. So lower levels of voltage scaling can greatly reduce static power.

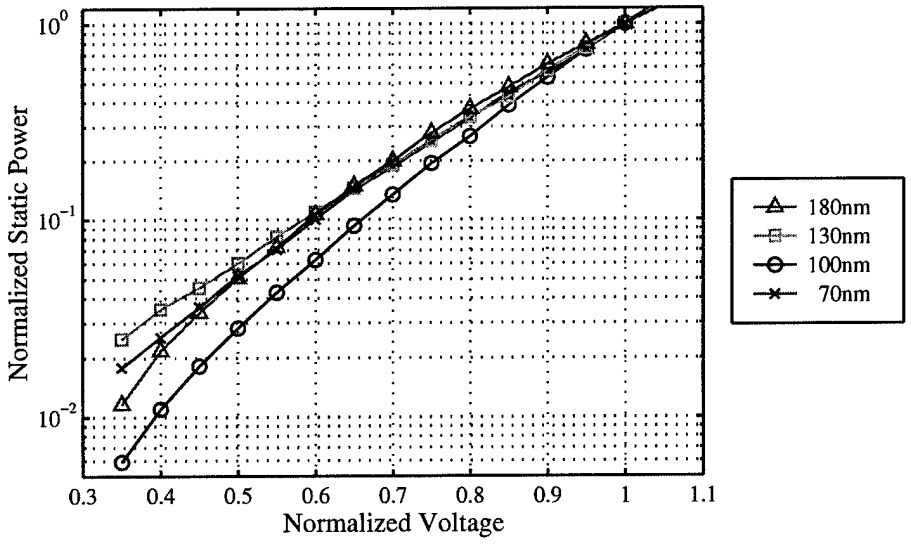


Figure 2.16. Pass Gate Static Power Dissipation

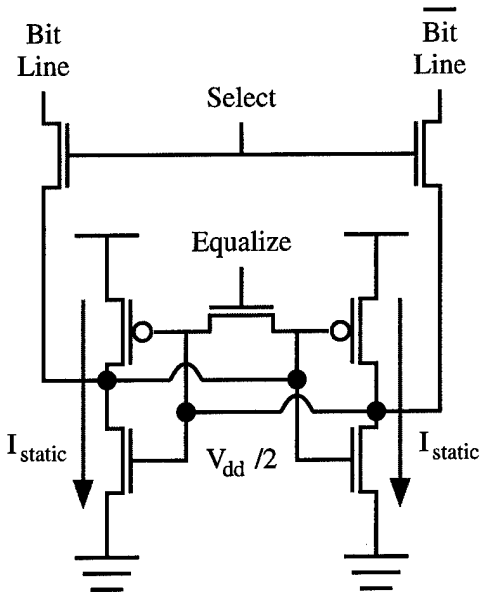


Figure 2.17. Sense Amplifier Circuit with Static Power Dissipation During Equalize Phase

As a second example, Figure 2.17 shows a sense amplifier circuit used typically in memory systems. During the *equalize* phase, the static power paths are composed of devices in the saturation region of operation. Equations 2.7 and 2.8 show the saturation drain current for long and short channel devices respectively [69]. Assuming that the devices are balanced, with an operating point of $V_{dd}/2$, leads to the power consumption shown in Equations 2.9 and 2.10.

$$I_{D\ long\ channel} = \frac{k'}{2}\frac{W}{L}(V_{GS} - V_t)^2(1 + \lambda V_{DS}),\ V_{DS} > V_{GS} - V_t \tag{2.7}$$

$$I_{D\ short\ channel} = \kappa \upsilon_{sat} C_{ox} W (V_{GS} - V_t),\ V_{DS} > (1 - \kappa)(V_{GS} - V_t) \tag{2.8}$$

$$P_{static\ long\ channel} = K(\frac{\lambda}{8}V_{dd}^4 + (\frac{1}{4} - \frac{\lambda V_t}{2})V_{dd}^3 + (\frac{\lambda V_t^2}{2} - V_t)V_{dd}^2 + V_t^2 V_{dd}) \tag{2.9}$$

$$P_{static\ short\ channel} = K(\frac{V_{dd}^2}{2} + V_t V_{dd}) \tag{2.10}$$

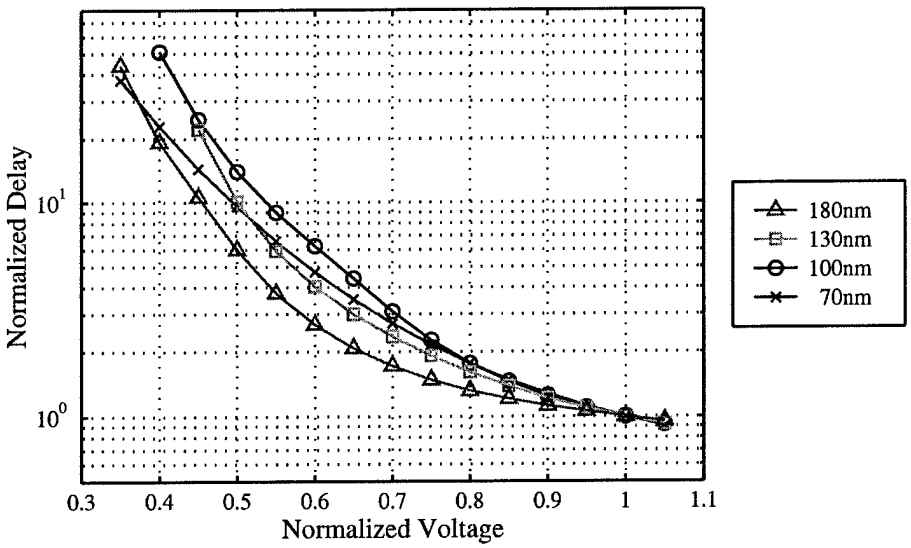


Figure 2.18. Delay of Sense Amplifier

For the sense amplifier, shown in Figure 2.17, voltage scaling is a bit more tenuous. Figure 2.18 shows the effects of voltage scaling on delay. Delay of this circuit is somewhat complicated, as it is a combination of tuned delays, including the word-line (not-shown), bit-line, select (not shown) and equalize signals. The relative delay of each of these signals is critical to the proper operation of the sense

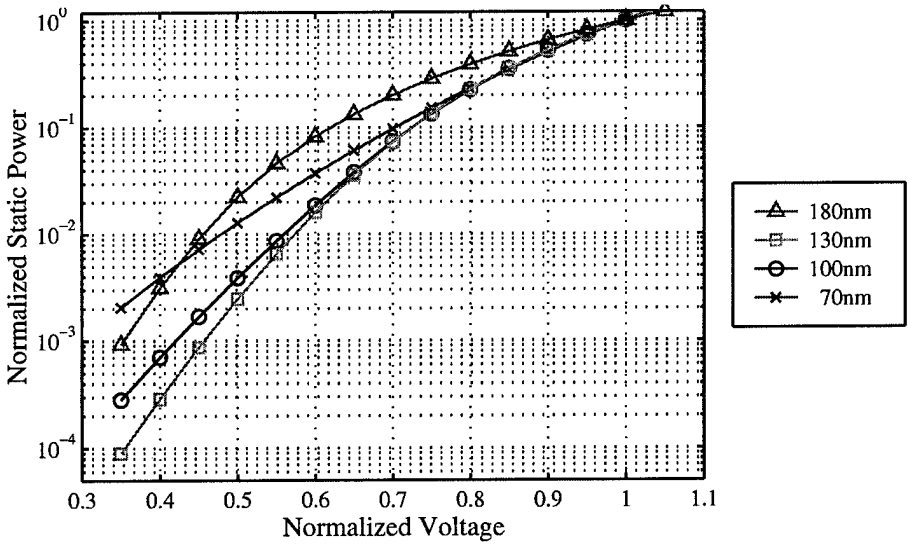


Figure 2.19. Sense Amplifier Static Power Dissipation

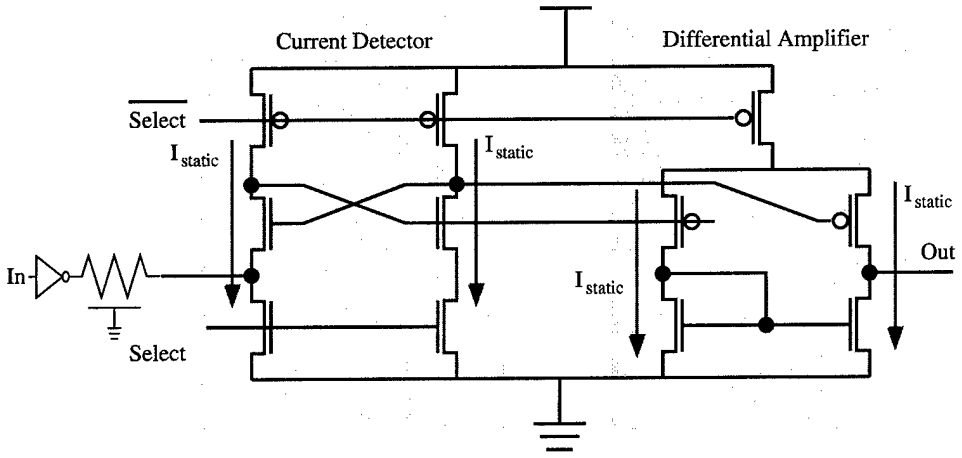


Figure 2.20. Single Ended Current-Mode Receiver [2]

amplifier. In general, the bit-lines must swing some amount before the select is enabled and equalize is disabled. Additionally the strength and operating point of the sense amplifier is voltage-dependent. These factors could make voltage scaling difficult. After some tuning, a circuit was developed, which could handle voltage scaling over almost the entire tested range. The minimum point in $180nm$ technology and the lowest two points in the $130nm$ technology did not function properly. The power results, in Figure 2.19, show a significant power savings for this type of circuit, specifically four orders of magnitude.

Current mode signal sensing, shown in Figure 2.20, with its two stages of analog amplifiers, is even more complicated. Successful operation of both the *current detector* and *differential amplifier* stages requires careful selection of operating point. Voltage scaling shifts these operating points, which eventually results in the circuit locking into a specific state. As a result, voltage scaling can only be applied over a limited range of voltages. The delay of these circuits, shown in Figure 2.21, includes one surprise. The $70nm$ system does not appear to be as adversely affected during voltage scaling as circuits in the other technologies. In this case, the normalization does a disservice, as the $70nm$ system does not function as quickly as expected for its maximum voltage. The leakage current in this technology adversely affects

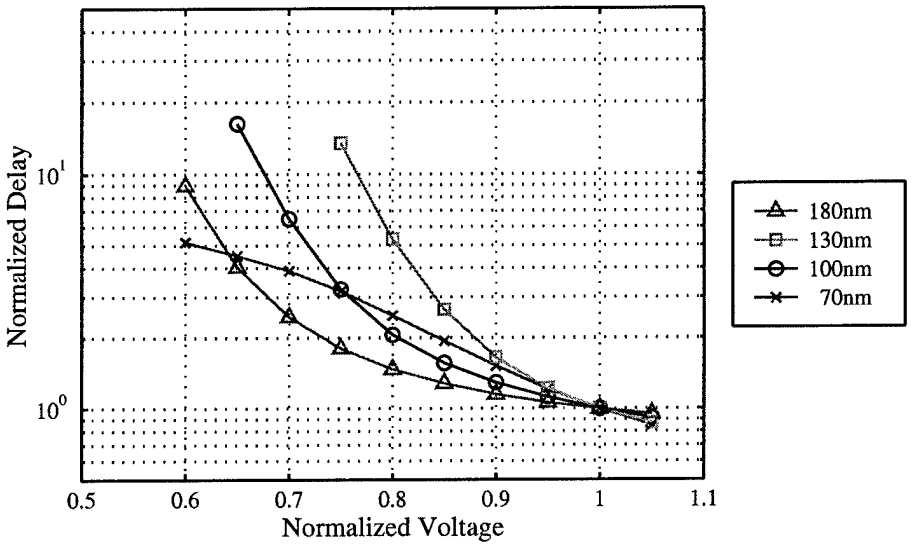


Figure 2.21. Delay Current Mode Interconnect

the system operating condition and speed. As the voltage scales down subthreshold leakage is reduced and the operating point improves. The transistors could be resized to improve high voltage performance but the data is included as an interesting side case of voltage scaling. Power reduction, shown in Figure 2.22, shows significant savings, similar to those observed in the sense amp system.

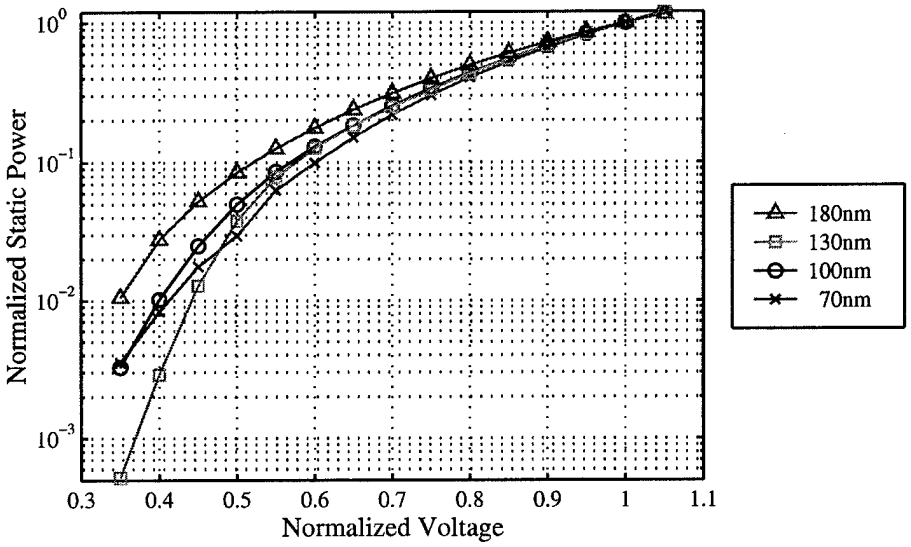


Figure 2.22. Normalized Static Power Current Mode Interconnect

2.3.5 Power Components Summary

In the final analysis, voltage scaling provides significant savings across all components of the power equation, Equation 2.1. The most impressive savings come in the short circuit and static power components. In general, however, the cost, in terms of delay, is even greater. The power-delay product actually increases by up to 500%. The worst delay increases came when introducing pass-transistors in the logic. Additionally, some circuit types failed to properly function over an entire range of scaling. All of this leads to the following conclusions about voltage scaling:

1. Voltage scaling should be applied when the clock can be made slower without affecting system performance. This implies slack exists in the system.

2. Careful design and testing must be accomplished on systems, which are intended to use voltage scaling. The transistor ssizes must be chosen so the circuit operates over the intended range of supply voltage scaling.

3. Critical path analysis must be completed over the entire range of intended scaling. The critical path network may change during scaling.

2.4 Voltage scaling

Assuming that the challenges presented in the previous section can be overcome, there are two distinct choices for voltage scaling: continuous DC-to-DC conversion or voltage selection. Intuitively, continuous conversion is more complicated but offers better potential for exploiting system slack. As such, a host of charge pump DC-to-DC converters have been developed [76, 77, 78, 79, 6, 80] to achieve high efficiency. The presentation in [77] provides convincing benefits for continuous, or nearly continuous, ranges of DC conversion. This paper shows the energy lost when discrete voltages are used for arbitrary clock rates, but does not address the timing overhead in switching or the system complexity added.

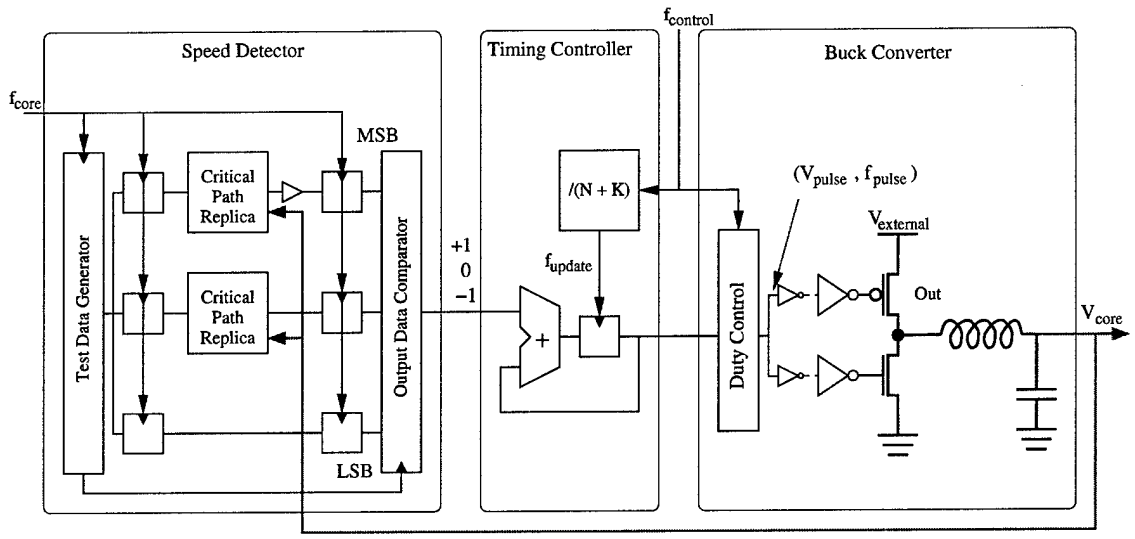


Figure 2.23. Variable Supply Using Buck Converter [6]

2.4.1 Charge Pump Implementation

The regulator scheme proposed by Kuroda et al. [6], shown in Figure 2.23, is especially elegant with its relatively simple digital control. Of the implementations, this presentation contains the most comprehensive set of design equations, making it the most repeatable and usable system. There are three major components of this system, from right to left: the *buck converter*, the *Timing Controller*, and the *Speed Detector*. The buck converter generates the output voltage and current for the core. In order to provide reliable regulation over a range of currents, the inductor and capacitor must be off-chip components. This will be demonstrated shortly, using the design equations. The output voltage is set by the duty cycle of V_{pulse}. The longer the low phase of V_{pulse}, the higher the output voltage. The frequency of V_{pulse}, f_{pulse}, helps set both the output voltage ripple and the efficiency of the converter, as shown in Equations 2.12 - 2.15. The duty cycle control compares a 6-bit count value to a programmable 6-bit threshold. Count values less then the threshold reset V_{pulse}, while those above set it. The counter is driven by $f_{control}$, which, in turn, is 64 times f_{pulse}.

The *Timing Controller* sets the rate at which the duty cycle threshold is updated. The control frequency, $f_{control}$, is reduced by $(N + K)$, where N is the maximum count value of the duty cycle counter, 64, and K is a factor, which has the effect of low pass filtering the threshold. The adder continuously adds the previous threshold with incremental changes sent from the *Speed Detector* system.

The *Speed Detector* system uses test data and critical path models to evaluate the present core voltage against the required core clock, f_{core}. The three paths in this block create a thermometer code, where a 001 is too slow, a 011 is correct and a 111 is too fast. This code is decoded to create the $+1$, 0 or -1 required by the adder in the *Timing Controller* to update the duty cycle threshold. This novel method eliminates the need for costly analog-to-digital converters in the feedback loop.

$$\eta = \frac{V_{core}I_{core}}{V_{core}I_{core} + I_{core}^2 R + P_{overshoot} + P_{control}} \tag{2.11}$$

$$P_{overshoot} = \frac{V_{external}^2}{R}\frac{\beta^2}{24} \tag{2.12}$$

$$P_{control} = P_{Speed\ Detector} + P_{Timing\ Controller} + P_{Duty\ Control} + P_{Buffers} \tag{2.13}$$

$$\beta \equiv \frac{T_{pulse}}{\sqrt{LC}} \tag{2.14}$$

One of the most important equations in the design process is the expression for efficiency, η, Equation 2.11. In this equation, power consumption is broken into four components: P_{core}, $P_{outputstage}$, $P_{overshoot}$, and $P_{control}$. The core power is simply $I_{core}V_{core}$ as shown, and the output stage power is a function of the IR drop across the output transistors. Overshoot power, Equation 2.12, is dependent on the ratio of f_{pulse} and the filter time constant as shown in Equation 2.14. To reduce this power consumption the filter time constant should be large, and the frequency,

f_{pulse}, should be fast. Having a fast pulse frequency, however, negatively impacts the power in the control circuitry, shown in Equation 2.13. The most important portion of this involves the buffers, which drive the output transistors, and could be thousands of times larger than minimum.

In addition to efficiency concerns, a power supply must meet several other key performance measures. One of the most critical is ripple, as shown in Equation 2.15. Once again the filter constant, β, plays a key role in limiting ripple. Additionally, Equation 2.16 shows the relative voltage drop for the system when the current, I_{core}, is applied suddenly. Obviously the LC filter must be designed to guarantee correct functionality under the worst-case conditions. Finally, for a variable voltage supply scheme the LC filter must be stable. The authors guarantee this by setting the filter damping constant, ξ to 1, as shown in Equation 2.17. This creates an over-damped response, which more slowly moves between initial and final value.

$$\frac{\Delta V_{core}}{V_{core}} = \frac{\beta^2}{16} \tag{2.15}$$

$$\frac{\delta V_{core}}{V_{core}} = \frac{I_{core}\sqrt{LC}}{CV_{core}} \tag{2.16}$$

$$\xi = \frac{R}{2}\sqrt{\frac{C}{L}} = 1 \tag{2.17}$$

While the buck converter typically achieves good regulation with high efficiency, it has not been used for on-chip dynamic voltage scaling outside of research initiatives. There are several key reasons for this. First, the LC filter is too large for on-chip devices. As such, additional off-chip pins are required for each regulation system. This prevents fine-grained application of DVS. Second, the large LC makes switching voltage levels relatively time consuming. The above system takes over $100\mu s$ to change voltages for the test chip described in the paper [6]. This represents more than 1000 cycles of overhead.

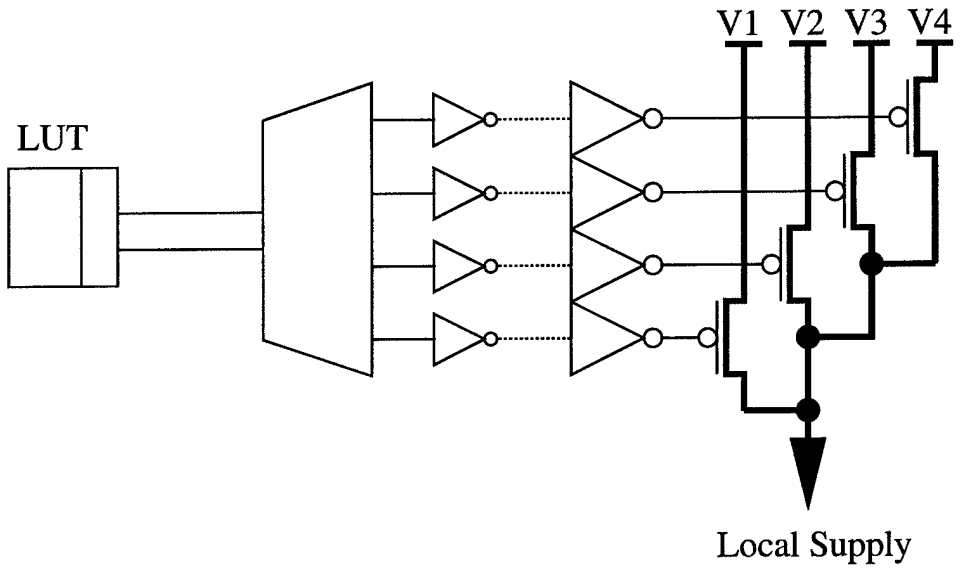


Figure 2.24. Variable Supply Using Voltage Selection

2.4.2 Voltage Selection Method

Future SoCs will contain hundreds of coarse-grained processing cores. As a result, reserving off-chip pins for each core will not be possible. Figure 2.24 shows a voltage selection scheme for voltage scaling of multiple processing cores. This method trades off voltage resolution for simplicity. StrongArm [81, 82, 59] allows dynamic voltage selection.

The efficiency, η, of the proposed voltage selection system is given in Equation 2.18. Note that $P_{control}$ in this equation is only leakage power when the system is not switching, as shown in Equation 2.19. The "ripple", in Equation 2.20, is partly based on the current drawn by the circuit and the ripple of the external supply. As such it is not a true ripple, but rather the voltage percentage by which V_{core} is reduced. And the voltage drop for sudden current changes is a function of the size of the pull-up transistors, as seen by the resistance, R, in Equation 2.21. The trade-off in this circuit come in sizing the pull up transistors. The larger the transistors are, the better the supply. Switching the supply may become more difficult if the transistors are made too small or too large.

$$\eta = \frac{V_{core} I_{core}}{V_{core} I_{core} + I_{core}^2 R + P_{control}} \tag{2.18}$$

$$P_{control} = P_{leak(Buffers)} \tag{2.19}$$

$$\frac{\triangle V_{core}}{V_{core}} = \frac{I_{core} R + \triangle V_{external}}{V_{core}} \tag{2.20}$$

$$\frac{\delta V_{core}}{V_{core}} = \frac{I_{core} R}{V_{core}} \tag{2.21}$$

2.4.3 Voltage Scaling Comparison

In both voltage scaling approaches, there is a need to calculate the size of the transistors in the output stage. To appropriately size these transistors the designer must know the maximum current expected by the core, $I_{core(max)}$, as well as the maximum voltage drop across the transistor, $V_{DS(max)}$. The first value, $I_{core(max)}$, is a parameter which must be measured, while the second value, $V_{DS(max)}$, is derived based on the required output resistance or power usage of the supply transistors. As a result, $V_{DS(max)}$ should typically be small compared to $V_{external}$, and the device will operate primarily in the triode region. Equation 2.22 shows how to calculate transistor width, w, for transistors in the triode region given $I_{core(max)}$ and $V_{DS(max)}$. Table 2.2 shows the k' values for the Berkeley Predictive Technology Models [68].

$$w = \frac{I_{core(max)} \times \frac{l}{k'}}{(V_{dd(max)} - V_t) \times V_{DS(max)} - V_{DS(max)}^2/2} \tag{2.22}$$

Typically the resulting transistor sizes are very large and must be driven by cascaded inverters. Equation 2.23 [6] can be used to find the optimal size and

Technology	NMOS	PMOS
$180nm$	0.000188	$6.82E-005$
$130nm$	0.000232	$7.43E-005$
$100nm$	0.000284	$8.38E-005$
$70nm$	0.000281	$9.36E-005$

Table 2.2. Values for k' Measured in HSPICE using BPTM[68].

Max Supply Voltage	$1.8V$
Max Supply Current	$0.1A$
Power Supply Capacitance	$600pF$
Max Core Frequency	$20MHz$
Max Ripple	2%

Table 2.3. Motion Estimation Core: Power Supply Requirements

numbers, N, for these inverters in the cascade. The output transistor width is w_N, and the sum of transistor widths in a minimum sized inverter is w_0. Using a scaling factor, x, of 4 for each stage gives the best results [6].

$$N = \frac{log(\frac{w_N}{w_0})}{log(x)} \tag{2.23}$$

The remaining question is how much power does the coarse-grained voltage selection system save in comparison to the fine tunability of the buck converter. To answer this question a voltage scalable system is tested based on the motion estimation core developed by P. Jain [23]. Table 2.3 shows the requirements for the motion estimation core. The maximum ripple voltage is simply selected.

The values in Table 2.3 are found when the system is processing motion vectors using the full search method, with search windows of 16 × 16 pixels. This system has the potential for dynamic reduction of search window size to 8 × 8 pixels. This decreases the number of computations required and increases the throughput of the system by a factor of 8/3 [23], making the system a good candidate for frequency

Low Supply Voltage	0.85V
Max Core Frequency	7.5MHz

Table 2.4. Motion Estimation Core: Low Power Supply Requirements

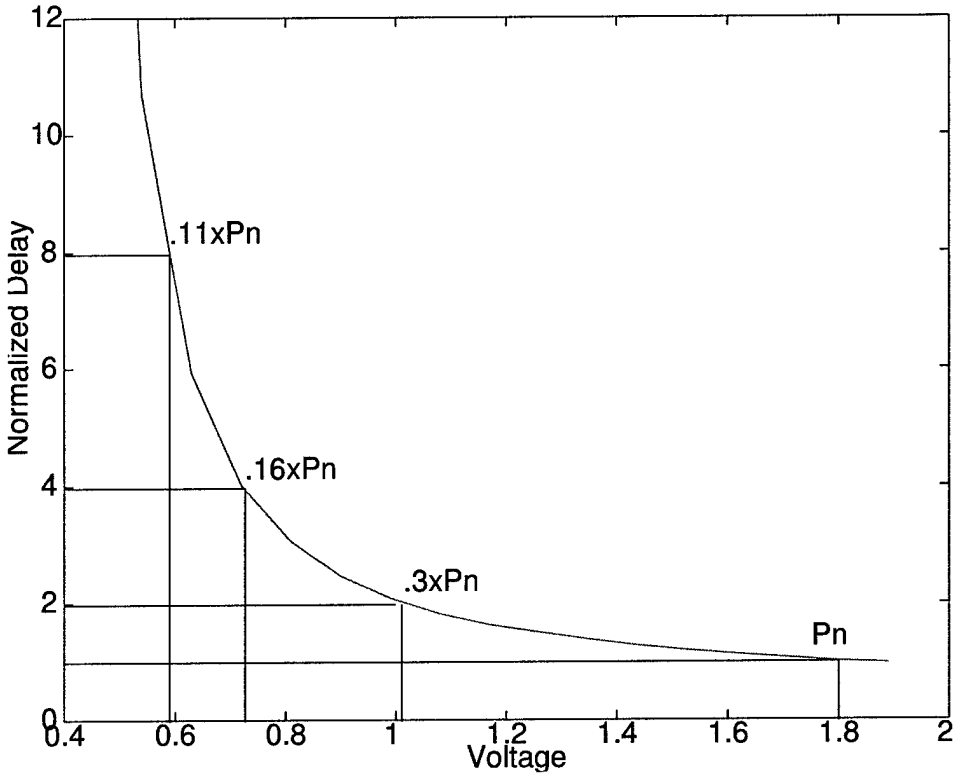


Figure 2.25. Variable Supply Using Voltage Selection

and voltage scaling. The resulting low voltage requirements are approximated in Table 2.4 using the voltage-delay properties of the ring oscillator shown in Figure 2.25. It is important to note that the voltage selection system requires discrete values of voltage which may not be representative of the core requirements. For simplicity, the discrete voltage is based on halving the core frequency, making the voltage $1V$.

Using the requirements in Table 2.3, power delivery systems are designed and tested using HSPICEl. Table 2.5 shows the results for the two systems.

	Buck Converter	Voltage Selector
Transistor Size	$1.2mm$ (NMOS) $5.3mm$ (PMOS)	$5.3mm$
Output Resistance	$0.37\ \Omega$	0.37Ω
Cascade Output Driver		
Cascade Stages	5 (NMOS) 6 (PMOS)	6
Scaling Factor per Stage	4.3 (NMOS) 4.3 (PMOS)	4.3
Ripple†	0.2%	2%
β	0.18	Not Applicable
L	$1\mu H$	Not Applicable
C	$30\mu F$	Not Applicable
Filter Time Constant	$5.6\mu s$	Not Applicable
$\delta V_{core}/V_{core}$	0.18%	2%
$V_{external}$	$2.0V$	$1.8V$
Core Power		
V_{core} = high ($1.8V$)	$180mW$ (max)	$180mW$ (max)
V_{core} = low ($.85V$ or $1V$)	$\approx 40mW$ (max)	$\approx 55mW$ (max)
Overhead Power $I^2_{core}R$ (high V_{core}) $I^2_{core}R$ (Low V_{core}) P_{VX} $P_{control}$	 $3.7mW$ (max) $0.82mW$ (max) $14.4mW$ $216\mu W$	 $3.7mW$ (max)) $1.12mW$ (max)) Not Applicable $9.5\mu W$
Efficiency High Voltage	90%	98%
Efficiency Low Voltage	62%	98%
Transition Time $1V$-$1.8V$	$\approx 5.6\mu s$	$< 2ns$

Table 2.5. Power Supply Comparison (for $.18\mu$ Technology)

Table 2.6 shows the resulting power consumptions for the high and low voltages. Even though the buck converter can be adjusted to better match the required voltage, the overhead causes additional power usage. This is primarily due to the overshoot power which does not scale with the core supply voltage. This overhead could be reduced at the cost of increasing the time the converter needs to switch between voltages.

System Power	Buck Converter	Voltage Selector
high V_{core}	$200mW$	$184mW$
Low V_{core}	$64.5mW$	$56mW$

Table 2.6. System Power Comparison (for .18μ Technology)

Chapter 3

Dynamic Parameterization

The computing capabilities in the modern era have enabled the development of highly parameterized algorithms. A parameter is any variable that can be modified to tune the algorithm for a specific task, environment, or set of user requirements. To meet the wide range of user requirements, digital signal processing (DSP) algorithms, including the MPEG standards [21, 22, 83, 84, 85], often contain many parameters, such as motion vector search algorithm and quantization matrix. As a result, these algorithms are called "highly parameterized". Additionally, as these algorithms rarely lead to specific hardware implementations, a host of other parameters, like architecture choice and pipeline depth, are introduced at this level of abstraction.

In the process of developing specific implementations, many parameters are typically fixed to reduce system complexity. With this in mind, however, the required flexibility of DSP applications often dictates some level of run-time reconfigurable parameterization. This creates *dynamically parameterized* implementations, which are defined by an ability to vary parameters at run-time. Once again MPEG [21, 22, 83, 84, 85] is a good example where, in order to view data, a decoder must be able to deal with varying frame sizes, bit rates, and even changes in how groups of frames are ordered. In addition to the required flexibility, the ever-decreasing cost of hardware makes it possible to incorporate additional parameterization to help a system optimize aspects of performance. The difficulty in parameterized systems is often choosing when or if parameters should be fixed when implementing an architecture. Although parameters can be bound at varying stages of the system's

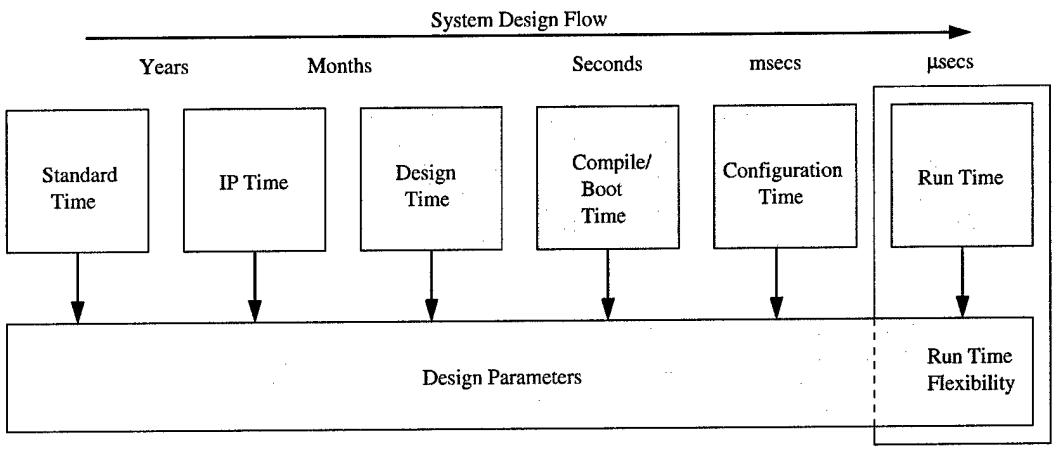


Figure 3.1. The Spectrum of Parameter Binding Times in a System Design Cycle

design cycle shown in Figure 3.1, it is the goal of this work to choose a selected subset of parameters to remain flexible or reconfigurable at run-time.

Achieving this goal is made difficult by the need to develop systems to control the run-time settings of reconfigurable parameters. Ideally each parameter should be set and changed autonomously at run-time to improve system performance. As such, a parameter must be linked to a metric which can be used to control the run-time reconfiguration. Metrics–like power consumption, system throughput, and data quality–are any system property which can be measured during operation. When developing the required control systems, metrics must be found which clearly indicate which parameter setting is best for the present operating conditions. For example, MPEG video encoding measures the usage of the output buffer as a metric to control quantization as a parameter. As the output buffer fills, the level of quantization is increased to reduce the required bit rate through the buffer and into the channel. In this case higher buffer usage directly correlates to a need to decrease bit rates. Furthermore, the control systems required in this process represent additional overhead, which can offset the performance gains of parameterization. To be effective, the cost of computing metrics must be low.

Dynamic parameterization is presented here as a formalized process for the development of adaptive computing systems. This approach first provides a methodology for choosing which application and architectural parameters are allowed to vary at run-time. It uses design of experiments [86] to construct and evaluate a configuration space for parameters. Carefully selected parameter combinations are simulated with both high-level and hardware-level software to identify best-case settings over the expected range of system operating conditions. Second, this approach evaluates the cost and effectiveness of metrics used to control the run-time selection of parameters. As in the MPEG example, the metric must be highly correlated to the operating condition it attempts to represent. At the same time, computing and using metrics should not significantly decrease performance.

Although dynamically parameterized systems can be developed for any aspect of performance, this dissertation focuses on power consumption. The approach for other performance factors, like latency and throughput, are analogous. It is important to note that the use of parameters in the development of power-aware systems is common. In fact, the term "parameterized" is so general that it covers all power-aware systems. As such, the use of parameters is not novel. The approach for parameter selection and control system development, however, is interesting in that it is the first to attempt to formally evaluate the entire design space.

3.1 Background

As stated in Chapter 2, there are many examples of systems which dynamically adjust their computing for reduced power. L. Benini presents a comprehensive view of this field in his book "Dynamic Power Management" [9]. In this book, L. Benini develops a base list of parameters useful in reducing power, as well as various approaches for varying these parameters dynamically. The base parameters

discussed include those described in Chapter 2: switching activity, clock frequency and supply voltage. Although his book gives detailed system examples, it does not describe an overarching approach for algorithmic parameter exploration and selection. Of the many power-aware publications, Bhardwaj et. al. [14, 15] presents possibly the most general approach in the development of "Point Systems." In this approach, a power "optimized" architecture is developed for each permutation of the given input parameters. During operation, the operating conditions dictated the selection of the appropriate architecture. This work does not fully explain the initial selection of parameters or the metrics used to vary them at run-time. "Approximate Signal Processing" [61] discusses a high level approach for evaluating trade-offs between output quality and performance. It uses "performance profiles" to map possible resource allocations to measures of output quality. Many other example systems have been developed [87, 23, 28, 29, 66, 67, 5]. In general, the parameterization of these systems is a result of an intimate understanding of the algorithm and architecture.

Much of this work has been driven by the demands of wireless communications, specifically the need to deal with varying channel properties. As a result, there are many dynamically parameterized architectures and algorithms for channel decoding, which adaptively change the complexity of the decoder according to channel signal to noise ratio [61, 88, 89, 90]. These wireless systems have critical power requirements and the opportunity to use reduced computation complexity for various applications or environmental conditions.

3.2 Approach

Dynamically parameterized systems, as shown in Figure 3.2, consist of two major components: the signal processor, and the parameter controller. Conceptually, the processor block consists of the algorithm, and the architecture on which it is

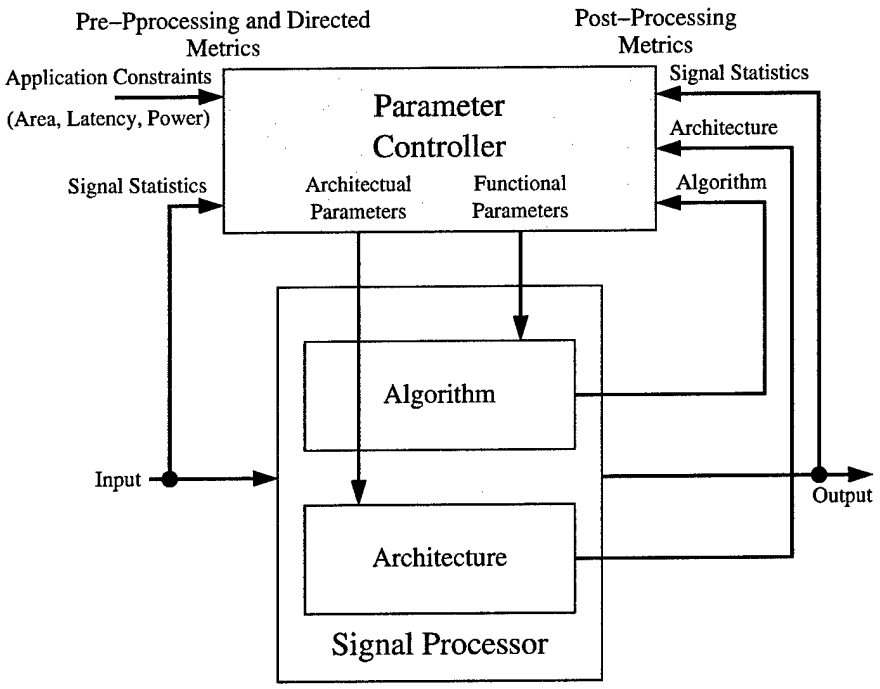


Figure 3.2. Dynamically Parameterized System Approach

implemented. As such, the parameters of the processing block can be classified as either algorithmic or architectural. Algorithmic parameters, like frame ordering and motion estimation search type in MPEG [21], vary the algorithm, and typically result in changing aspects of the output data. Architectural parameters, like word size and pipeline depth, do not change the underlying computation, but may affect resulting accuracy, speed, and power consumption. The controller modifies these parameters at run-time based on the evaluation of a set of metrics. These metrics represent commands and any variable that can be measured or inferred from the operating environment and actual operation of the system. In a broad sense there are three classes of metrics.

1. *Directed metrics*, including power and throughput limits, consist of system requirements and constraints set by the user or compiler. These requirements can be specific flags, like "use **8 bit** mode", or more general thresholds, like "power limit = 2W".

2. Analyzing the input signals generates *pre-processing metrics.* Patterns, statistics, or even explicit flags in the input data can tell the controller the best processing approach.

3. *Post-processing metrics* are the result of monitoring the system and its output. These metrics can be used to predict the best parameters for future computation.

When reduced power consumption is a goal, the optimization problem can be stated as shown in Equation 3.1 - 3.2.

Minimize:

$$Power = \Phi(\mathbf{m}) \tag{3.1}$$

subject to:

$$\dot{\mathbf{m}} = f(\mathbf{m}, \mathbf{p}, t) \tag{3.2}$$

where the vector of metrics, $\mathbf{m}$, represent the state of the system and vector of parameters, p, represent the control. Power is a function of system state, and the state is a differential equation of previous states and control.

Although theoretically sound, this approach is not very practical due to the large number of parameters and the lack of state functions. At the onset of the design problem, the number of the parameters and metrics may be unknown. For most applications and architectures there is a multitude of parameters and metrics to choose. Also, it is usually unclear how the metrics and parameters interact with each other and how they affect power. Dynamic parameterization, as shown in Figure 3.3, is a heuristic approach to evaluating and selecting parameters and metrics in the development of a dynamically parameterized system. Given the baseline algorithm, exploration of previous research and standards can provide and help refine an initial list of parameters and metrics. These sources may also provide

baseline architectures and software for the algorithm. The first phase of dynamic parameterization is *software simulation.* In this phase there are actually two sets of simulations: one to evaluate the parameter list and another to evaluate possible metrics. These two are interlocked, however, as parameter evaluation may lead to the inclusion or elimination of metrics. Metric evaluation may do the same to the parameter list. As such, the process is somewhat iterative, with parameters and metrics being added or eliminated in each iteration.

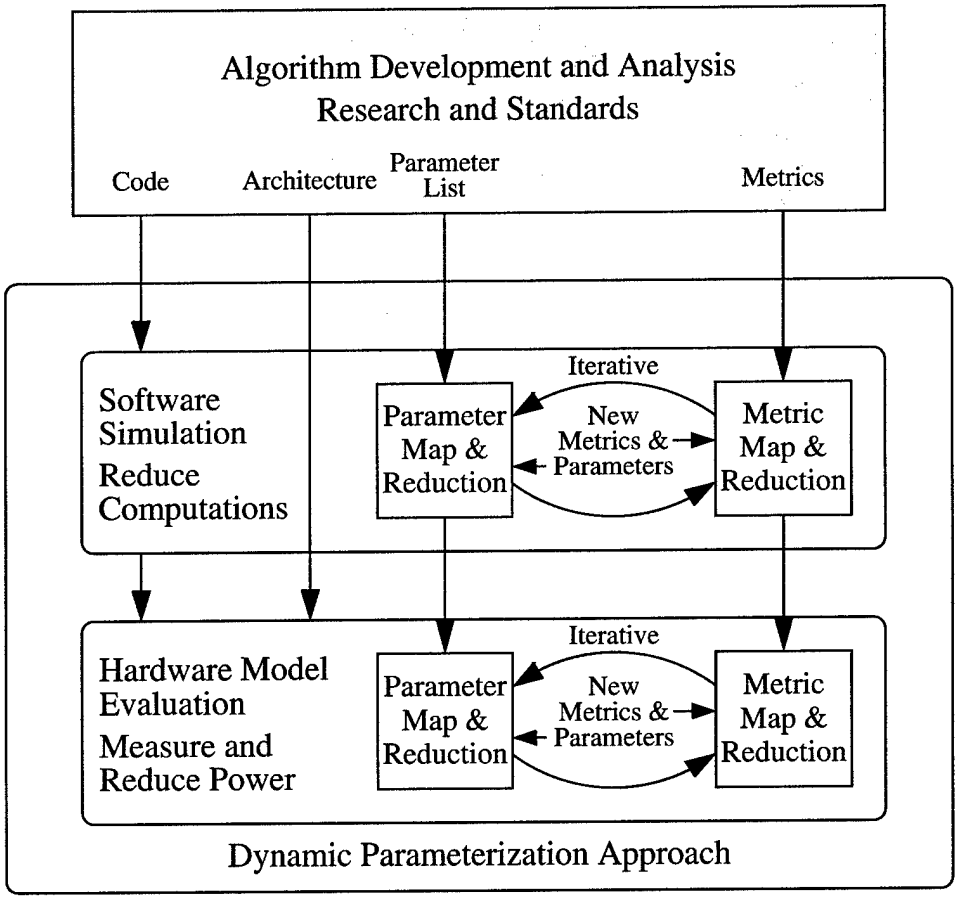
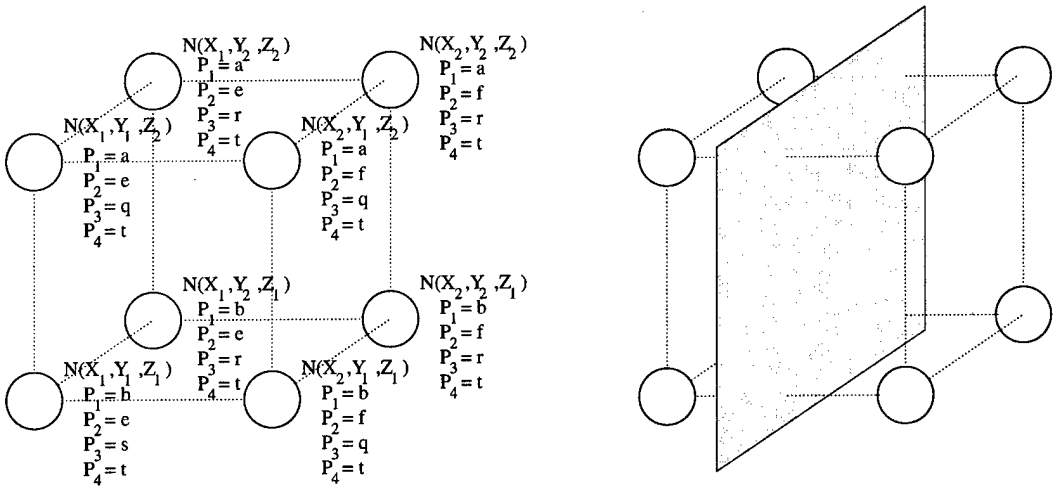


Figure 3.3. Dynamic Parameterization Approach

In the parameter map and reduction block, a design of experiments [86] approach is used to evaluate how each parameter affects instruction count. Design of experiments provides a formal method for evaluating large numbers of parameters over

a wide range of input data and environment characteristics. First, the input data and environmental characteristics, which form the environment space of the design problem, are discretized by considering combinations of extremes. If necessary, design of experiments does allow for evaluation of midpoints, but the key is to reduce a possibly continuous multi-variable space to something more manageable. In the same manner, the parameter space is discretized. Fortunately, given the digital nature of the problem, this process is typically trivial. Each parameter value is evaluated independently at each node in the environment map for its effects on instruction count. The result is a set of nodes in the environment space, which are characterized by a list of parameter settings, which result in the lowest instruction count. Nodes in the space with the same parameter settings can be grouped, and parameter settings that do not characterize any node can be removed. Additionally, the decision to allow runtime variation of a given parameter must be made by weighing the costs in system complexity against the potential benefit of adaptation. It is important to limit the extent of adaptation to only those parameters with the best possible performance trade-offs. As such, parameters, which do not significantly impact instruction count, can be removed from consideration. Of course care must be taken in this stage not to over-prune the parameter lists, as instruction count may not correlate with the power consumption of the final implementation. An example design space is shown in Figure 3.4.

Using the same discretized environment map, the metric map and reduction block attempts to identify metrics, which clearly delineate nodes. While at first glance the metrics seem obvious, care must be taken as the controller, which evaluates them, represents additional overhead for a dynamically parameterized system. As such, excessive controller calculations can outweigh the benefit of system adaptation. For example, theoretically we expect the environment for video encoding to include frames with both high and low motion content. It is pointless to measure motion as



a. Parameter Map of Design Space

b. Metric Dissection of Design Space

Figure 3.4. Example Parameterized Configuration Space and Metric Partition

a metric, as the controller would then be performing the very calculation it hopes to control. The key is to extract the information necessary to identify the operating point, with a minimum amount of processing. For this, the best metrics may be environmental statistics, which are highly correlated indicators of specific power-performance trade-offs. Figure 3.4(b) shows how metrics may divide a parameterized design space.

After evaluation of the design space at the software level the system must be studied at the hardware level with a reasonably accurate model of the target architecture(s). In this stage, the reduced parameter and metric lists are evaluated in the same way as before on the representative hardware. Architectur-specific parameters and metrics may be added to the lists for consideration. Controller structures should be developed to accurately evaluate their costs. The end result is a dynamically parameterized architecture that includes the controller necessary to adjust the selected parameters at run-time. It should be noted that the resulting system uses discrete parameterization. This is done to reduce the complexity of the design space, but it also may lead to sub-optimal results. Increasing the resolution

of specific variables, including supply voltage and frequency, may be advantageous if high-resolution metrics can be obtained.

3.3 Dynamically Parameterized Architecture Example: Motion Estimation

Most of the applications targeted by MPEG video standards use lossy coding techniques to meet specified storage and transmission requirements. An important compression technique reduces temporal redundancies by transmitting only the difference between consecutive frames. This technique can be enhanced if the images in the frames can be aligned to minimize the overall difference. In this case, the information is coded as a frame difference and a series of associated alignments, called *motion vectors.* These motion vectors represent the movement of image components from one frame to the next. A motion estimation process is used to find these motion vectors.

Motion estimation compares a current frame with a previous or sometimes a future search frame. The current frame is divided into macro blocks of 16×16 pixels. Each macro block in the current frame is compared against a region in the search frame, referred to as a search window. The coordinates of the best matching block in the search frame become the motion vector for the block under consideration.

The compression aspect of motion estimation reduces the number of bits sent through the rest of the system. Fixed bandwidth systems, which use such a lossy compression scheme inherently trade quality for compression. Thus, compression ratio achieved during motion estimation directly impacts video quality in fixed bandwidth systems.

3.3.1 Parameterized Motion Estimation Architecture

To analyze the design space, parameter flexibility is designed into a pipelined motion estimation architecture, shown in figure 3.5 [23]. Although independently designed, this matrix-based architecture is similar to the GA-2D systolic array

designed in [91]. Both architectures use a matrix of processing elements to compute the absolute difference of pixels between the two blocks. This base architecture contains a pipeline, designed primarily for speed, and can evaluate a series of 352×240 frames at 30 frames a second in full search mode at 106MHz. In addition to the processing array, the architecture includes an address generator unit (AGU), which selects how to pull data from memory, and input FIFOs to arrange the data and set up the pipeline. The block in the current frame, the current block, is stored in the array one pixel per processing element, and the search pixels are input every clock cycle. The differences calculated at each element are passed down the array and added to eventually compute the sum of absolute differences (SAD) for all the pixels in a current block against a search block.

This architecture allows variation of the following parameters:

- *Algorithms*: Full search (FSA), 3-step search (TSS) and Spiral search Algorithm[91]
- *Search Window Size in FSA*: 32×32, 16×16, 8×8
- *SAD Threshold for Spiral Search*: 2762, 3840, 7943
- *Pel Subsampling in FSA and TSS*: None, 2:1, 4:1
- *Pixel Width in FSA and TSS*: 8-bit, 4-bit, 1-bit

All search algorithms use the processing element array and differ only in the addresses of the search blocks fetched. The AGU implements a state machine to generate the appropriate memory access patterns for the different search methods and sizes. The FIFOs store and share these fetched pixels to reduce external memory accesses when possible. The SAD thresholding is implemented at the end of the pipeline and signals the AGU to test the next block. Both Pel Subsampling and Pixel Width variation are implemented in the processing array.

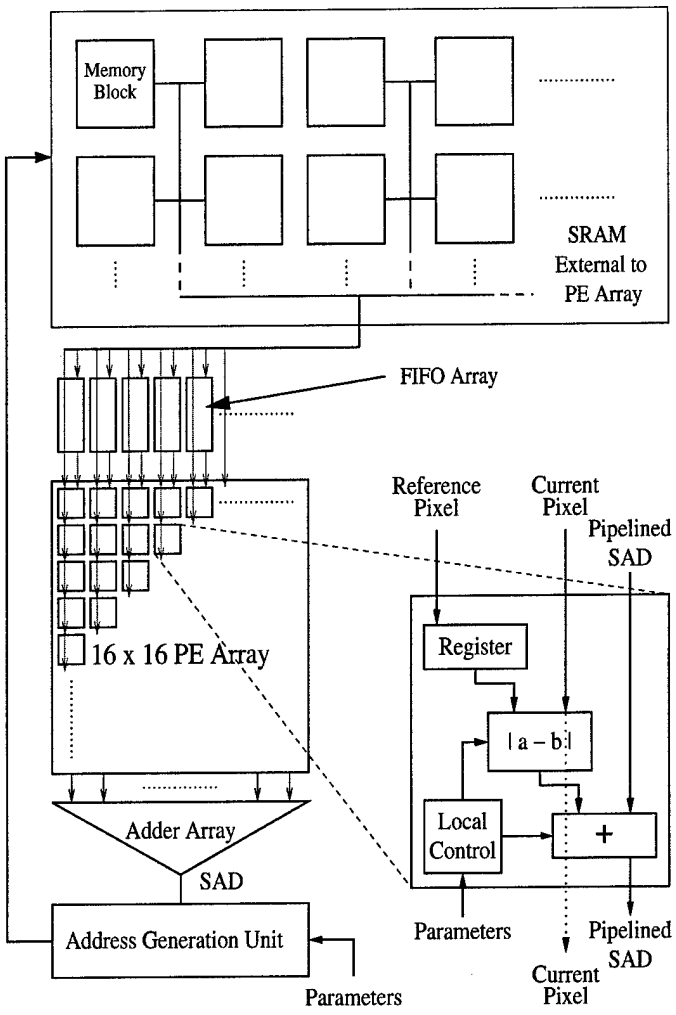


Figure 3.5. Parameterized Motion Estimation Architecture

All circuit components except the memory are synthesized with a commercial .18μ standard cell library, and evaluated using Synopsys RTL Power Estimator. This tool takes the input design, the technology file and our video stimulus to calculate system switching activity and the associated power. Power for the unimplemented memory is approximated by counting accesses generated by the AGU. Memory power ranges from 14% of the total power during full search, when the FIFOs can most effectively share data, to 43% in three-step search, where the data for each search block must come directly from memory. The input/output overheads of the bus were not evaluated. The motion estimation architecture can be used in both

Parameters	Matsushita	Toshiba	ME 2001
Power Consumption	90mW *	240 mW *	30 mW - 1W
Frame Size	176x144	176x144	352x240
Frame Rate	15 fps	15 fps	30 fps
Clock Frequency	54 MHz	60MHz	110 MHz
Algorithms	n.a.	n.a.	FSA, TSS, Spiral
Process	1.8μ	2.5μ	0.18μ
Vdd	1.8V	2.5V	1.8V

* Power numbers include components not associated with motion estimation, including audio processing.
Motion estimation numbers and search algorithm are not available, n.a.

Table 3.1. Architecture Comparison with Commercially Available Devices

MPEG-2 and MPEG-4 standard encoders. The present implementation, however, does not include an MPEG-4 alpha-plane and no parameters were tested that affect this portion of the MPEG-4 standard.

At this stage it is important to recall that the test architecture has been developed to evaluate the design space, not as a final low-power implementation. As such, it incorporates more flexibility than will be needed or even effective in a final system. In spite of this limitation, the test architecture, including reconfiguration overhead and the inefficiencies of standard cell implementation, has comparable performance specifications to recent implementations by Toshiba [50] and Matsushita [49]. Table 3.1 compares the three implementations.

It is hoped that the structures and results of this parameter space analysis can be used to develop array-based MPEG motion estimation architectures. Clearly, parameter variations, which impact the size of the search space, will alter the power consumption for any implementation. The relative impact of these parameters against those of Pel Subsampling and Pixel Width, however, are only reasonable in similar array-based architectures. In addition, the results are valid for technology scaling as long as leakage power does not dominate.

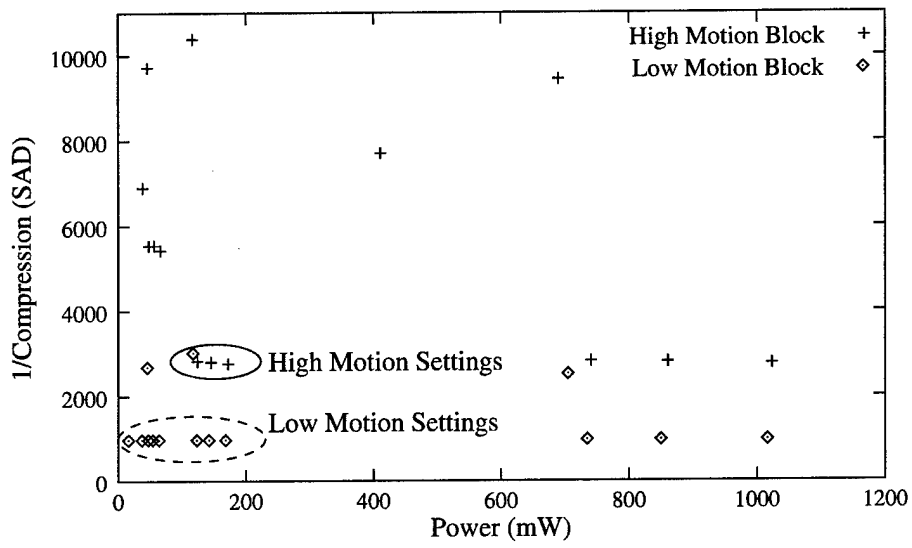


Figure 3.6. Parameter Summary Versus SAD Value

3.3.2 Configuration Sample Space

Figure 3.6 combines the results for the various parameters tested. In general, the best operating conditions are represented by points close to the origin with both low-power consumption and low SAD values. The configuration points within the solid oval represent the best operating conditions for high motion blocks, and are characterized by a full search algorithm with a search window size of 16×16 pixels. Similarly, the dashed oval contains configuration points that provide low-power and high compression for low motion blocks. The points are characterized by a full search algorithm with window sizes of 8×8 and 16×16 pixels. The 3-step and the spiral search techniques also work well with low-motion blocks and are included in the dashed oval.

3.3.3 Controller Metric Evaluation

The previous parameter analysis indicates that search window size most clearly characterizes the operating conditions best suited for analyzing blocks with high and low motion. This section attempts to find a metric, which can be used to select one

of two possible search window sizes: large, 16×16; or small, 8×8. The first step involves finding a way to identify the amount of motion expected in each block so the controller can select, in the second step, the most power-efficient window size for processing. In order to perform this analysis, a simple signal statistic must be found to accurately predict the amount of motion without introducing significant controller overhead. Three possible motion-predictor methods were tested for correlation with motion vector magnitude.

1. The SAD value of current block and search block with same coordinates: For this correlation method the controller computes the SAD value for collocated blocks in the current and search frames. If this value is larger than a specified threshold, it is assumed that the motion vector also will be large. This method introduces an overhead of 330 block SAD calculations per 352×240 pixel frame. This analysis must be performed prior to motion estimation using the existing array of processing elements, leading to reduced pipeline throughput. Additional memory space and data accesses are needed to accommodate these SAD values.

2. Pixel contrast in the current block (peak-to-peak difference): The contrast in a block is represented by the difference between its highest and lowest luminance values. When the contrast value is high, the block is likely to have high frequency components, which may make the matching process more difficult. As a result, a larger search window may be more appropriate. The controller must calculate the contrast of the current block before attempting to find the motion vectors. The controller would have to contain this contrast hardware. Pixel contrast can be performed in parallel with motion estimation and will incur no memory overhead.

3. Motion vectors from the previous frame: The controller, to predict the motion in the current frame, uses motion vectors from the previous frame. A simple threshold determines processing with a large or small window size. The only other overhead is additional memory space and data accesses.

To test the correlation of each of the possible predictor methods, four video sequences were processed: *table tennis*, *football*, *flower garden*, and *mobile*. The motion vectors of each frame were found using the immediate predecessor as the reference. Figure 3.7 shows the correlation between the candidate predictor magnitude and current motion vectors. To make this comparison, the pixel contrast and SAD predictor magnitudes were scaled to match the range of magnitudes found in the motion vectors. A candidate predictor is correlated if its value can be used to predict the magnitude of the current motion vectors within 5 pixels. Clearly, the previous motion vectors, with a total of 90% correlation, provide the best prediction method with the least overhead.

To complete this analysis, the predictive motion vector magnitude at which the system would switch from a large to a small search window size was determined. A simple threshold technique is used and our adaptive system is tested with the four sets of video data. When a predictive vector is larger than the threshold, the 16×16 pixel window is used. Those predictive vectors smaller than the threshold trigger use the 8×8 pixel window.

Figure 3.8 shows two sets of data. First, it shows the percentage of blocks that find the minimum SAD value for a given trigger threshold value. The actual minimum was found by searching the entire frame. Second, it shows the percentage of blocks that use the smaller search window size for the different trigger thresholds. As this trigger point approaches zero, more blocks use the large window size. As a result of this larger window, more blocks find the minimum SAD. When the trigger point gets larger, more blocks fail to find the minimum SAD. Our data indicates

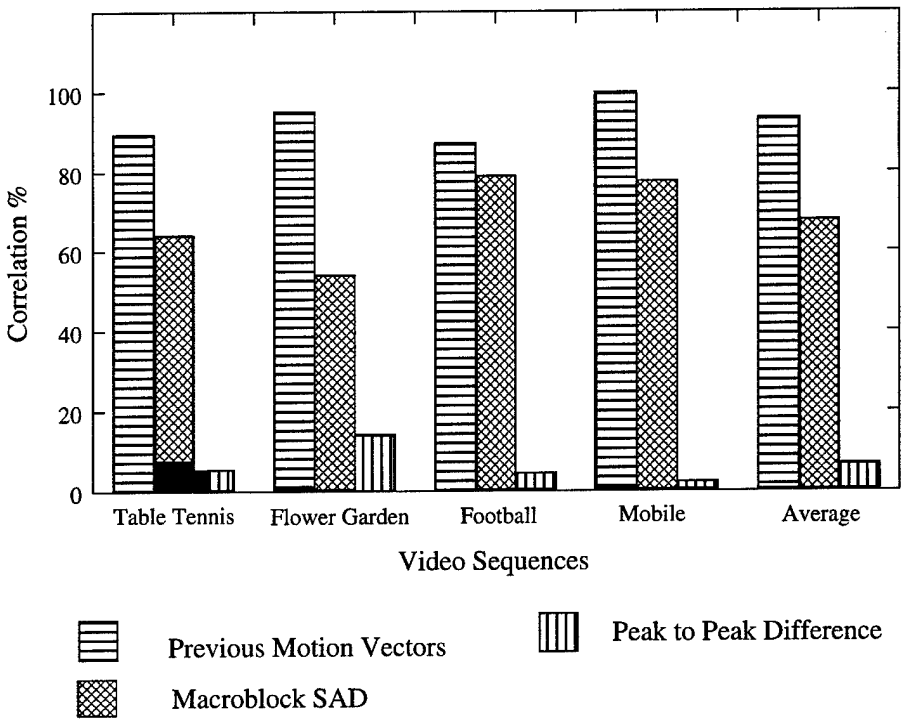


Figure 3.7. Predictor Correlation with Current Motion Vectors

that when all motion vectors are calculated using the large window, 10% of the blocks fail to find the minimum SAD. This indicates that an even larger window is required for 10% of the blocks. Figure 3.8 shows that setting the threshold at 3 reduces the percentage of minimum SAD found by less than 1%. At the same time, this threshold enables the motion estimation architecture to calculate more than 70% of the vectors using the smaller search window. Using the 62% power savings of the 8×8 pixel window (as shown in section 3.3.2), this system saves over 40% of the power of a static 16×16 pixel search window system. In this approximation, the predicted vector magnitude and trigger point comparison operations are assumed to consume little power. They are calculated only once per block in contrast to the 65k additions required to find the motion vectors in an 8×8 search window.

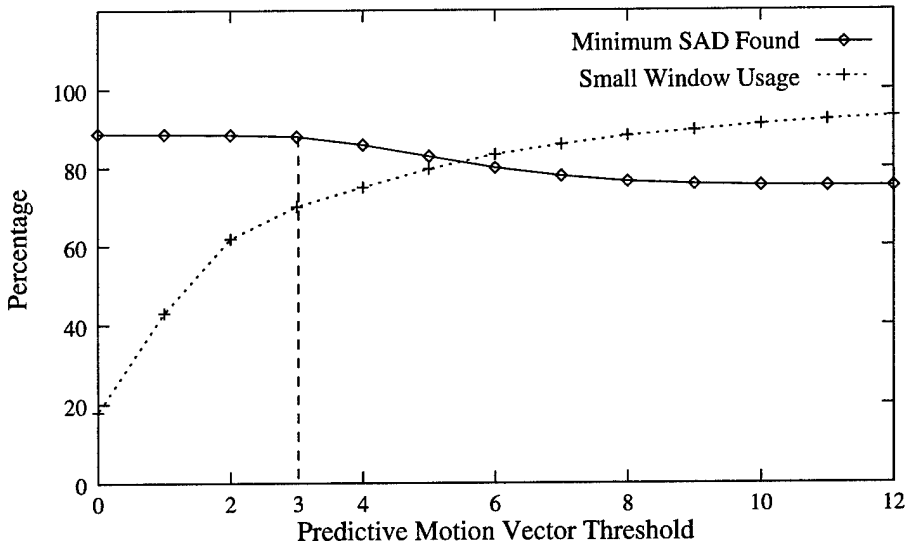


Figure 3.8. Search Window Selection and Resulting Performance for Variations in Predictive Motion Vector Threshold

3.4 Other Computations

The design methodology applied to motion estimation has also been applied to four other DSP applications: discrete cosine transform [24], Lempel-Ziv compression [92], 3-D graphics [28], and adaptive Veterbi decoding [89]. Although these systems are explained in detail in the original documents, some of the details are reproduced here to further demonstrate the applicability of dynamic parameterization. In each application the researcher developed a configurable application specific architecture to explore the configuration space. As a result, all of these applications could be used as dynamically parameterized cores in the adaptive system-on-a-chip (aSoC) implemented in this work.

3.4.1 Discrete Cosine Transform

The two-dimensional discrete cosine transform (DCT) [93] is an integral part of many image and video compression systems. The DCT design described in [67] uses dynamic parameterization in the row-column classification (RCC) power-saving

feature. RCC dynamically adjusts the number of arithmetic computations per calculation based on signal properties measured at an early stage in the pipeline. This adaptive technique shows a 35-40% power savings for a full custom implementation. A soft core DCT design [24] recently has been implemented to allow further design space exploration. Power for this design was determined with the Synopsys Power Estimator. Table 3.2 shows that a power benefit exists for RCC in soft core implementations.

Test Bench (Std. Dev.) (8x8 Pel Matrix)	Power (mW)		Power Savings
	NO RCC	With RCC	
Football Block587 (10.3)	743.430	633.532	14.78%
Football Block3 (61.9)	823.253	648.354	21.26%
Football Block1048 (97.8)	828.546	651.234	21.41%
Mobile Block496 (1052.1)	843.054	660.010	21.71%
Garden Block745 (2458.1)	843.736	660.675	21.69%
Tennis Block236 (2762.4)	826.153	655.954	20.60%
Mobile Block3 (7184.2)	801.535	654.584	18.33%
Football Block1297 (8602.2)	818.183	661.096	19.19%

Table 3.2. RCC Power Savings Impact for a Set of Natural Images

3.4.2 Lempel-Ziv Compression

Lempel-Ziv (LZ) compression is a loss-less compression technique that is used in a wide variety of communication and storage applications. The algorithm used to implement Lempel-Ziv compression represents a large class of computations, which rely on variable length matching sequences (e.g. bio-sequence matching, data mining). The parameters for the LZ algorithm can be set depending on input data statistics and system power and compression constraints.

The fine-grain parallelism of LZ compression has been exploited in a variety of recent systolic array and CAM implementations [92]. The LZ algorithm has two

main parameters, which can be dynamically configured: 1) the longest matching length, and 2) the dictionary or sliding window length. Longest matching length can easily be tracked and used to modify the matching length parameter in the compression hardware. The size of the dictionary (sliding window length) can also be modified dynamically by tracking the LZ pointers to determine how frequently remote sections of the dictionary result in matches. Each of these parameters affects the compression ratio and speed. As a result, applications that can withstand a varying compression ratio could save power. Figure 3.9 shows how a small network is affected by load and compression ratio [94]. It is clear that the required compression ratio depends on the network load.

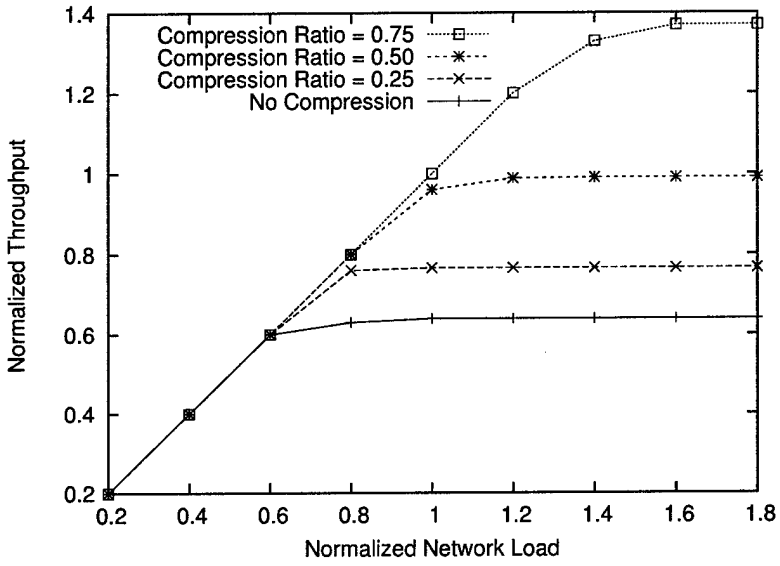


Figure 3.9. The Effect of the Mean Compression Ratio on a Network of 10 Nodes with Probability of Bit Error = 1.0e-5

3.4.3 3D Graphics Light Rendering

Real-time 3D graphics will be a major contributor to power consumption in future portable embedded systems. Fortunately, we can exploit content variation

and human visual perception to significantly reduce the power consumption of many aspects of 3D graphics rendering. In [28] we study the impact of novel adaptive Gouraud and Phong shading algorithms on power consumption. The adaptive algorithms exploit graphics content (e.g. motion, scene change) and human visual perception to achieve low power operation without noticeable quality degradation. Novel dynamically configurable architectures are proposed to efficiently implement the adaptive algorithms in power-aware systems with gracefully degradable quality.

There are two variable parameters considered in the 3D graphics architecture: shading algorithms and specular computation. Exploiting visual sensitivity to motion, Gouraud or Phong shading algorithm is selected, depending on the speed of an object and its distance from the camera. The same selection criteria is also used to select the type of specular computation to be used.

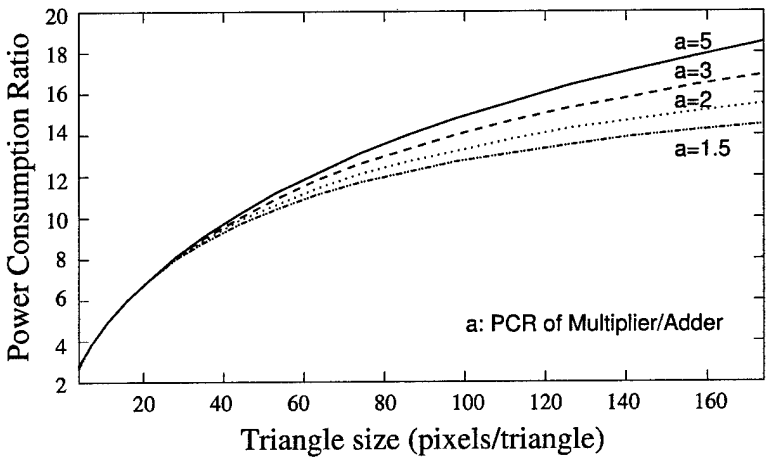


Figure 3.10. Power Consumption Ratio of Phong and Gouraud Shading: One Triangle Shading

Figure 3.10 shows that power consumption between Phong and Gouraud shading varies by a factor of 20 for large triangles. In situations where human visual perception permits, using the lower quality shading algorithm (Gouraud) can save

significant power. Results based on simulations using short but realistic rendering sequences indicate power savings up to 85%.

3.4.4 Adaptive Viterbi Decoding

Convolutional codes, which allow for efficient soft-decision decoding, are widely employed in wireless communication systems. As convolutional codes become more powerful, the complexity of the corresponding decoders generally increases. The Viterbi algorithm (VA) [95, 96], which is the most extensively employed decoding algorithm for convolutional codes, works well for codes with short constraint length K. For more powerful codes with large constraint lengths the adaptive viterbi algorithm (AVA) [88, 90] is used. It reduces the average number of computations per decoded information bit. Our work looks at using AVA to achieve reduced power consumption.

There are two dynamic parameters used in the architecture built for this work : constraint length and truncation length. The constraint length indicates the number of times each input bit has an effect on producing output bits [89]. A trellis diagram is used to determine the most likely transmitted data bits. The number of time steps used to identify the most likely transmitted symbol sequence is called the "truncation length".

These parameters vary depending on the noise levels in a channel and the bit error rate (BER) requirement of the system. Table 3.3 shows a comparison of constraint lengths for various ranges of channel signal-to-noise ratio (SNR). For a large amount of channel noise, the constraint length must be large to achieve a low BER, but under low-noise conditions, it can be kept small. Table 3.3 demonstrates the speed advantages of this trade-off, but power savings could potentially also be achieved for constant decode rates.

K	FPGA decode (Kbps)	Decode rate w/PCI overhead (Kbps)	Max. FPGA clock (MHz)	SNR range (dB)
4	333.7	186.0	40.5	6.3-6.5
5	164.2	117.7	20.1	6.1-6.3
6	162.3	116.3	19.9	5.5-6.1
7	160.8	114.2	19.7	3.9-5.5
8	143.6	109.4	17.6	3.7-3.9
9	141.1	107.8	17.3	3.1-3.7
10	101.5	NA	25.5	3.0-3.1
12	94.8	NA	24.7	2.8-3.0
14	82.3	NA	23.0	2.5-2.8

Table 3.3. Decode Rate Versus K for XC4036XL-08 (K = 4 to 9) and XCV1000-04[97] (K = 10 to 14)

3.5 Summary

Table 3.4 presents a qualitative summary of the computational parameters and corresponding observed trade-offs for the five applications discussed in this chapter [20]. It shows how each of the tested parameters changes performance and cost factors in the applications. Most importantly for this document, this table shows that the formal method of dynamic parameterization is applicable to many application specific architectures. Using this careful and structured evaluation of the system configuration space provides a parameter map which indicates which parameters should remain flexible in the final implementation. The analysis also provides a formal method for evaluating and selecting the metrics needed for run-time control of parameters.

Computation	Parameters	Range	Trade-offs			
			Performance		Cost	
			Latency	Compression or Quality	Area	Power
Motion Estimation	Algorithms	Full Search	Degrades	Improves	-	Degrades
		TSS	Improves	Degrades	-	Improves
		Spiral	Good for Small Searches	SAD Threshold Dependent	-	Good for Small Searches
	Search Window Size	1 x 1 Pixels to Complete Frame	Degrades	Improves	-	Degrades
	Pel Subsampling	Psub 2:1	Degrades	Slight Degradation	-	Improves
		Psub 4:1	Degrades	Slight Degradation	-	Improves
	Bit-Width	1-8 Bits	-	Improves	-	Degrades
	SAD Threshold	0 to 65280	Improves	Degrades	-	Improves
DCT	MSBR	Bit Variation	Degrades	Improves	-	Degrades
	RCC	1 to 4 Clk Cycles	Degrades	Improves	-	Degrades
	RAC	1 or 2 Units	Improves	-	Degrades	Improves Vdd scaling
Lempel-Ziv	Matching Length	1-2048	Degrades	Improves	Degrades	Degrades
	Sliding Window Length	256, 512, 1024, 2048	-	Improves	Degrades	Degrades
3D Graphics	Shading Algorithms	Phong	Degrades	Improves	Degrades	Improves
		Gouraud	Improves	Degrades	Improves	Improves
	Specular Computation	Exponential	Degrades	Improves	-	Degrades
		Iterative Multiplication	Improves	Degrades	-	Improves
Adaptive Viterbi	Constraint Length	3 to 14	Degrades	Improves	Degrades	Degrades
	Truncation Length	9, 18, 27, 36, 45	Degrades	Improves	Degrades	Degrades

Table 3.4. Dynamically Reconfigurable Parameters and Trade-offs

Chapter 4

Scalable Reconfigurable System Architectures

At present there are many reconfigurable architectures that attempt to provide solutions to future computing needs and constraints. This chapter attempts to bound the design space and clarify terminology. As such, an extensive list of example systems is used to understand the many design trade-offs. A qualitative overview of modern reconfigurable architectures is used to show why aSoC is chosen as the demonstration system for this dissertation. In this discussion, aSoC is shown to represent a subset of reconfigurable systems likely to meet the future needs for special-purpose computing. Additionally, a detailed overview of the aSoC architecture reveals its potential for dynamic parameterization and voltage scaling. Understanding the base-line architecture is critical to the discussion of the added power-aware features in Chapters 5 and 6.

4.1 Taxonomy of Reconfigurable Architectures

Modern reconfigurable architectures can be classified on at least two dimensions: *processing element* type or *network* type. Both classifications give insight into the capabilities and limitations of the architecture. Additionally, target applications can often be inferred from the architecture selection.

4.1.1 Processing Element-Based Taxonomy

Figure 4.1 shows a qualitative perspective of the design space for the type of reconfigurable processing elements in modern architectures. With the vast number

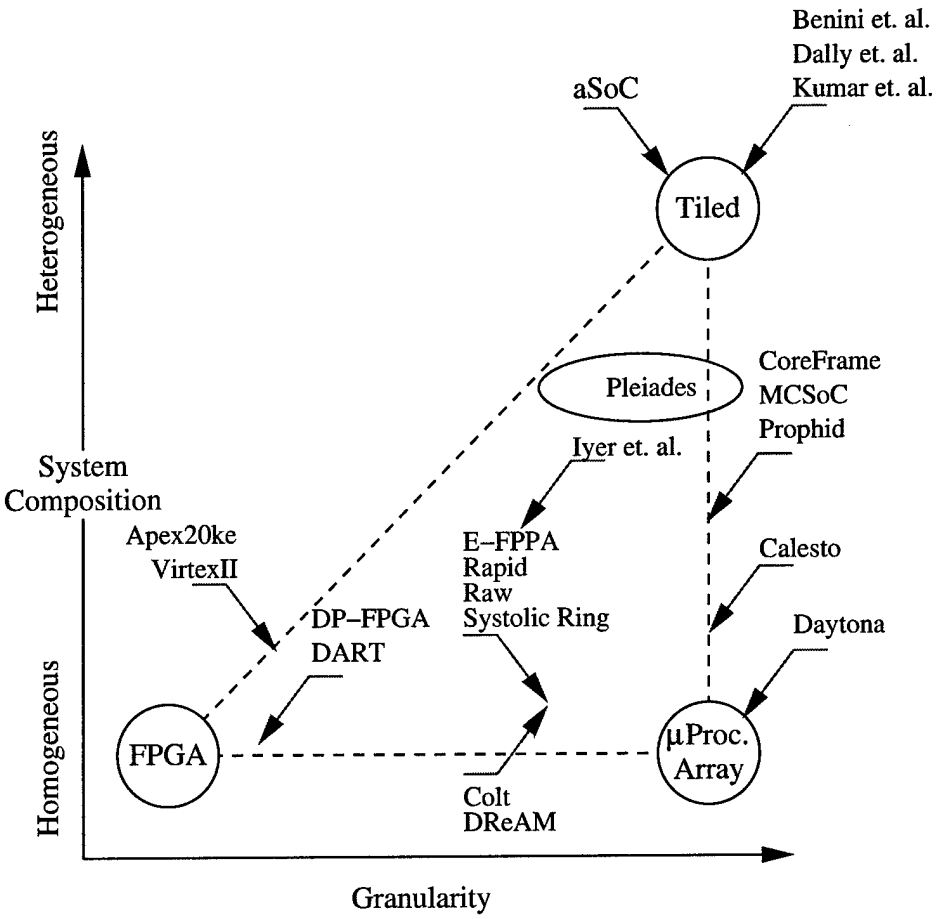


Figure 4.1. Reconfigurable Architecture Processing Element Design Space

of architectures proposed in both academic research and industry, it is not possible to describe or compare these systems in detail in this document. Therefore, Figure 4.1 is used to understand generalities in the design space and to identify features common to many systems. With this in mind, Figure 4.1 represents the design space with two axes. The horizontal axis, *Granularity*, is related to the size and processing power for individual processing elements in the architecture. The vertical *System Composition* axis represents the level of diversity allowed.

As shown in Figure 4.1, there is a huge continuum of reconfigurable architectures. At one end, multi-processor systems, like Lucent's Daytona Chip [98], represent the most coarsely constructed systems. In this case, the Daytona Chip simply

contains four, 64-bit processors [98]. From this point in the design space, various processor-centric reconfigurable architectures can be listed. A slightly more heterogeneous system, CALISTO from Broadcom [99], connects an array of digital signal processors (DSPs) to a reduced instruction set computer (RISC) to create a powerful signal-processing engine. More common than the strict multi-processor systems are those systems which use excess die area to add support circuitry for a central processing unit (CPU). Although the exact relationship of processing elements varies, systems like CoreFrame [100] from IBM, MCSoC [101], Prophid [102, 103] and various others [104, 105, 106, 107, 108] complement a processor with various peripheral devices. These processor-centric devices are often termed system-on-a-chip (SoC). The Pleiades concept [62, 63] and the Maia processor [109] are special cases of processor-centric reconfigurable architectures, as they allow higher numbers and more varied "satellite" systems including a field programmable gate array (FPGA). Finally, Iyer et al. [110, 111, 112] move processor-centric SoC to finer granularity in dividing a processor architecture into asynchronous subsystems. This allows for the application of power-reduction techniques.

Fine-grained cluster-based FPGA systems, like those discussed in [113], can be considered the other end of the reconfigurable architecture spectrum. DART [114], DP-FPGA [115], CHESS Array [116], and MATRIX [117] use fine-grained arithmetic logic units (ALUs) in place of look-up-tables (LUTs). The following long list of systems all use some type of a coarser-grained array of homogeneous processing elements connected with various interconnect strategies: DReAM [30], Systolic Ring [118, 119, 120], Colt [121], PADDI [122, 123], D-Fabrix [124], RaPiD [125], ECLIPSE [126], PipeRench [127], KressArray [128, 129, 130], Garp [131], FPFA [132, 133], REMARC [134], MorphSys [135] and RAW [136, 137, 138, 139, 140, 141]. These systems all resemble their FPGA ancestors in the primarily homogeneous nature of their processing fabric.

FPGA-like architectures have been developed for more specialized tasks through the incorporation of heterogeneous components. Xilinx's Virtex II contains multipliers and memory to better target DSP applications [72], while Altera's Apex20 device contains programmable logic arrays (PLA) that are suitable for high fan-in control structures [71]. Again Pleiades [62, 63], the Maia processor [109], CS2000 [142], and FIPSOC [143] bridge the gap between processor-centric reconfigurable SoCs and FPGA-like architectures by including both types of resources.

Finally, at the top of Figure 4.1 are completely heterogeneous, intellectual property (IP) core-based systems. aSoC [34] present a practical approach to integrating heterogeneous IP cores, while Beneni et al. [144, 145, 146], Dally [32], and Kumar [33] present more forward-looking views of reconfigurable SoC design. These systems will be discussed in more detail in Chapter 6.

This perspective of reconfigurable architectures reveals three types of approaches: processor-centric, FPGA-like and IP core-based, each with the potential for homogeneous or heterogeneous processing elements. In general, the more homogeneous the system, the wider the range of computing applications. As heterogeneous components often perform specific functions, their addition reduces the generality of the architecture. The architectures at the top of Figure 4.1 are interesting in that prior to IP core population they could be used for any problem. Once application-specific cores are placed in the network, the types of applications the SoC can effectively handle is greatly reduced. As such, aSoC represents a set of systems, which can be populated with IP cores to solve specific computing problems. The coarse granularity of these cores will reduce the development effort in creating application-specific systems.

4.1.2 Interconnect-Based Taxonomy

Understanding the choices for reconfigurable interconnection is complicated by the large variety of approaches. Figures 4.2 and 4.3 attempt to bound the problem graphically in terms of interconnect flexibility and overhead, respectively. As with the proceeding discussion on processing elements, these graphs attempt to provide a qualitative overview of the design trade-offs. This caveat is even more important here, as the terms "flexibility" and "overhead" represent composites of system properties. For this discussion consider the flexibility of an interconnect structure as the ease in which it allows arbitrary connectivity. Overhead is a qualitative combination of physical complexity and communication delay. The vertical axis in both figures is a qualitative measure of the scalability of these systems. For this discussion, scalability is a combination of the performance and feasibility of each architecture as they grow.

As shown starting in the lower left corner of Figures 4.2 and 4.3, some early reconfigurable architectures have been developed with dedicated or fixed interconnect buses. These include commercial methodologies like Coral [106, 147] and academic approaches as proposed by Hu et al. [148] or used in the globally asynchronous locally synchronous (GALS) radio [149, 150]. While Coral [106, 147] and Hu et al. [148] attempt to establish connectivity for arbitrary designs, the GALS implementation is especially attractive for smaller systems, which inherently form a pipeline [151]. These systems offer the potential for optimum communication bandwidth between communicating devices. Unfortunately, as the number of cores and interconnectivity increase, the physical design of dedicated interconnect becomes prohibitive. Routing dedicated interconnect from element to element becomes increasingly costly and difficult on the system floorplan. Additionally, uncertainty in final wire delays can make early performance evaluations difficult. As a result, dedicated routing is not expected to play a role in large-scale design.

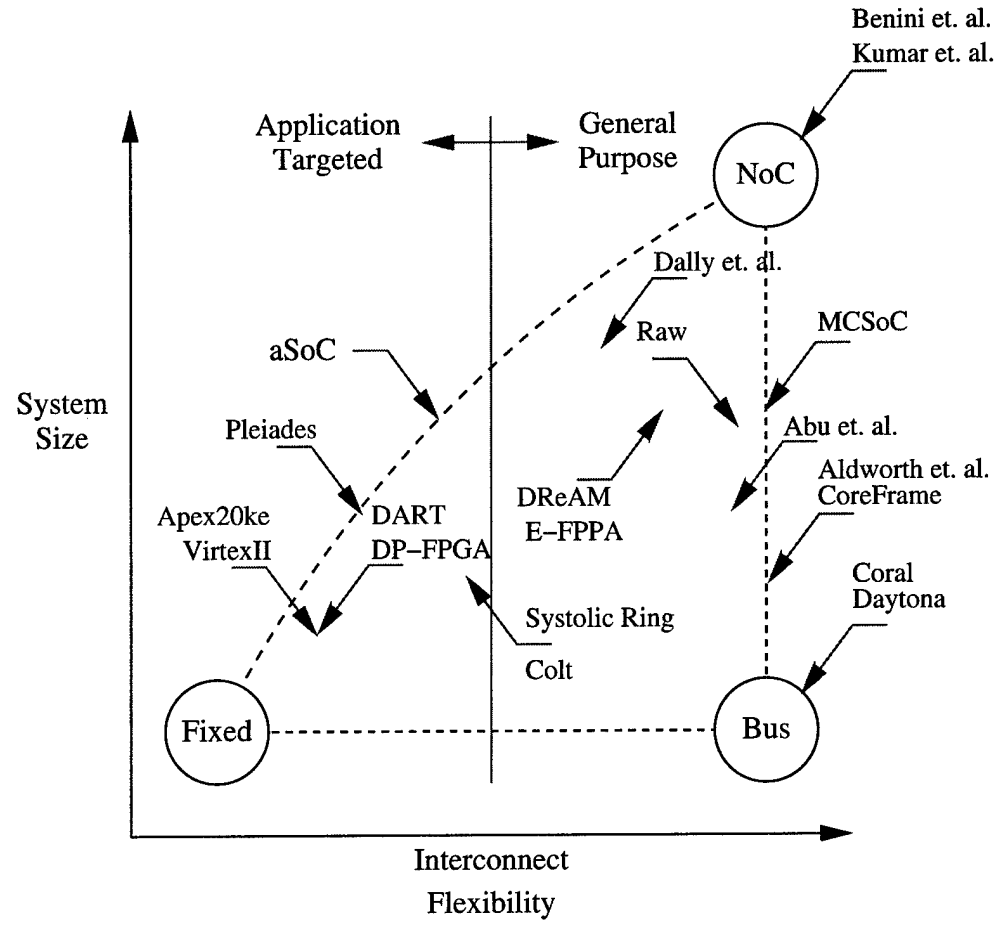


Figure 4.2. Reconfigurable Architecture Interconnect Design Space

To cope with some of these issues, a wide variety of arbitrated bus-based architectures proposed in both academia [101, 152, 153] and industry [98, 154, 108, 155] are represented in the lower right of Figures 4.2 and 4.3. Arbitrated bus-based systems often allow full connectivity of the processing cores by time-sharing a single set of wires. The interconnect delays and placement costs are predictable, and well-defined protocols exist for communicating over the bus [156]. This eases the verification effort. Arbitrated bus-based systems, however, have the limitation of allowing only one data communication on the interconnect at any given time. The bandwidth of the bus is divided across all the required communications. This could represent a system bottleneck as the number of processing elements increases.

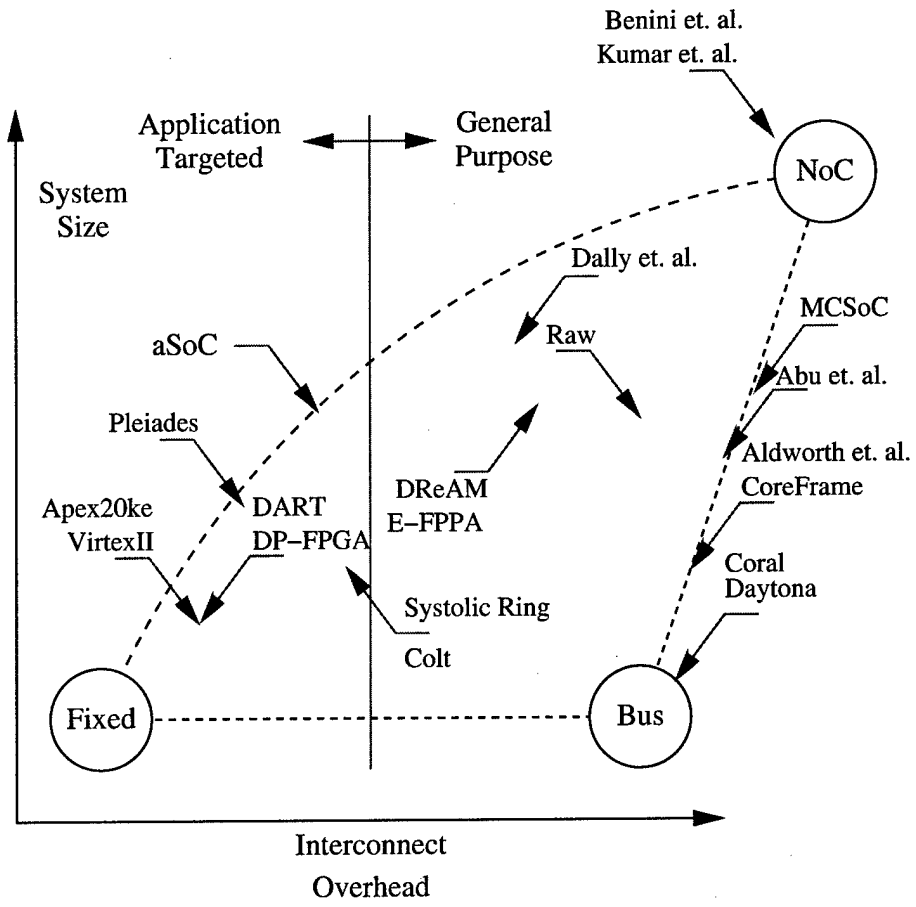


Figure 4.3. Reconfigurable Architecture Interconnect Design Space

For the single arbitrated bus, performance is the limiting factor to system scalability. To overcome this limitation, more complicated bus structures have been introduced [100, 105], and are conceptually represented along the right side of the charts. Multi-bus systems connected with bridges, including AMBA [157], have been proposed to overcome the scalability problems of the bus architecture. Data can be transmitted in parallel provided it stays on the local bus in the system hierarchy. This allows for full system connectivity, while increasing bus bandwidth. As a result, this increased performance extends the usefulness of bus protocols and standards.

Opposite the arbitrated bus-based systems are reconfigurable architectures, which use fine-grained routing segments. At the lowest level of flexibility, FPGA-like systems [158, 115, 124, 125, 116] set routing connectivity at compile-time. The

major benefit is the ability to use the routing as pipelines of various bit widths. This parallelism can help these systems achieve high performance for special applications.

In an attempt to improve performance and flexibility, many other systems use rings [159, 118, 119, 120] or mesh-based [143, 126, 128, 129, 130, 132, 133] structures as alternatives to arbitrated bus or FPGA-like interconnect systems. Some systems use heterogeneous interconnect strategies to provide both the performance of reconfigurable interconnect and the flexibility of bus-based systems [30, 142, 102, 103].

At the pinnacle of the charts is the SoC paradigm termed network-on-a-chip NoC [144]. Although nearly all SoC interconnect strategies could be called networks, the NoC concept has some key features that separate it from the approaches discussed this far. First, any network can be used on a chip. Second, within the network an asynchronous protocol is used to transfer data between cores with different clock domains. This concept specifically eliminates global clocking and potentially provides the best solution for applications requiring hundreds of cores with full and unpredictable communications. The structure of the network, however, is still limited to some degree by the two-dimensional nature of the chip. In addition, asynchronous routers and core wrappers increase the area overhead and network delay.

Although NoC represents the pinnacle of large-scale reconfigurable architectures, for some applications it may be overkill. In DSP applications, like MPEG [21, 22], the communication patterns may be known prior to system development. Although the communication patterns may be too complex for the development of dedicated interconnect, a reconfigurable interconnect structure can be used to create a pipeline. ASoC specifically targets these applications by trading some communication flexibility to reduce system overhead and potentially improve performance. In this respect, aSoC shares interconnect performance traits of FPGA-like reconfigurable architectures.

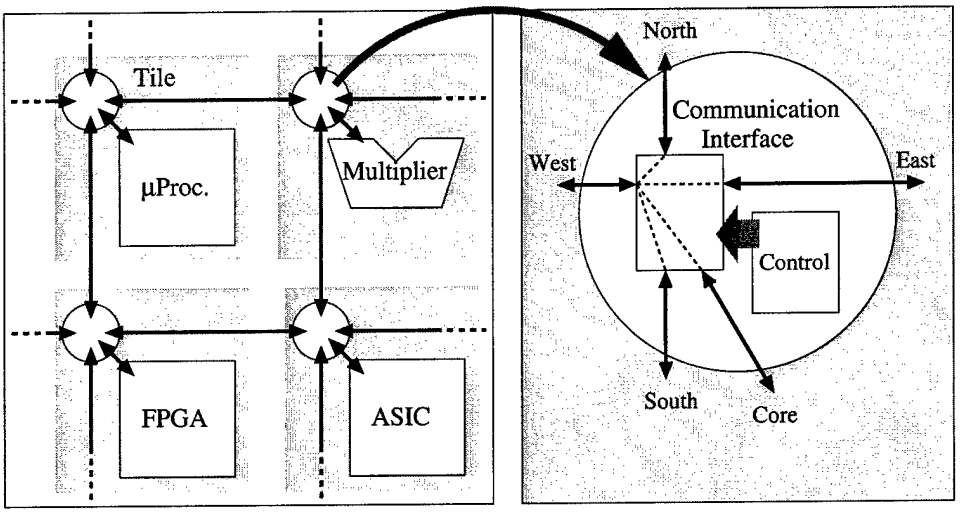


Figure 4.4. aSoC Architecture

4.2 Adaptive System-on-a-Chip, aSoC

As seen in preceding sections, aSoC represents a class of reconfigurable architectures well suited for large-scale, application-specific computing problems. The aSoC approach to interconnection is a scalable, mesh-based communication architecture that facilitates high bandwidth communication for integrated heterogeneous cores, as shown in Figure 4.4. IP cores are arrayed in a tiled floorplan and connected to the mesh with a *communication interface* capable of transmitting and receiving data to and from its four nearest neighbors. A standardized communication structure and simple interface protocol creates architecture modularity and provides a convenient framework for the use and reuse of intellectual property (IP) cores. By limiting inter-core communication to short wires with predictable performance, high-speed communication can be achieved. A novel aspect of this new architecture is the ability to vary the allocation of static and dynamic bandwidth on a per-application basis based on communication needs. With this ability, aSoC can handle the complicated and possibly time-varying communication patterns of large DSP applications.

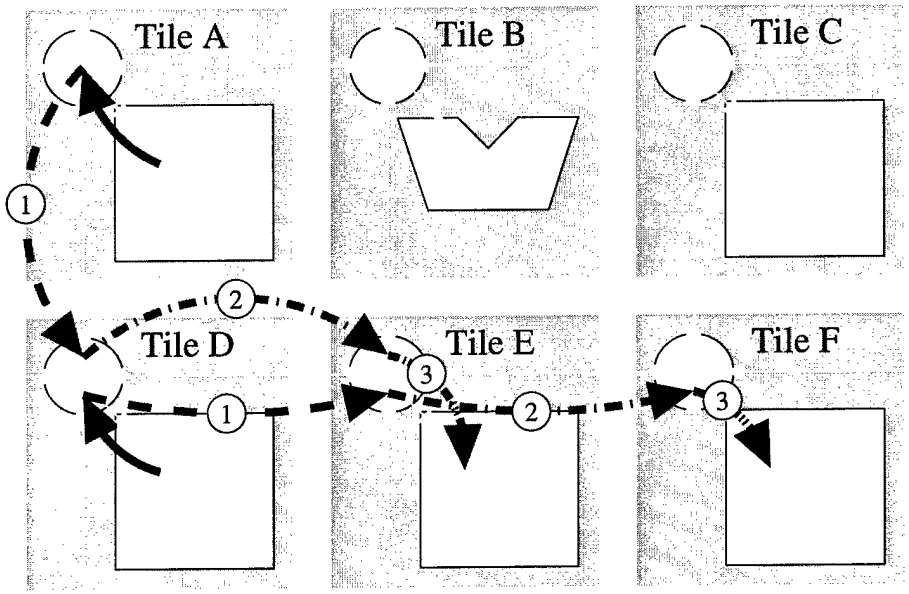


Figure 4.5. Pipelined Stream Communication

4.2.1 aSoC Communication Protocol

To be effective, an on-chip interconnect must be flexible enough to adapt to a range of applications. A significant amount of on-chip communication in next-generation SoCs will be consumed by high-bandwidth, stream-based data related to multimedia applications. The primary representation of data transfer in aSoC involves the use of data *streams*. A stream is a pipelined connection between a source and a destination core. Data transmission on each pipeline stage, be it a core to its core interface or between neighboring core interfaces, requires a single communication clock cycle.

Two example data streams are shown in Figure 4.5. One stream starts in *Tile D* and ends at *Tile F*, while the other goes from *Tile A* to *Tile E*. For the first stream, data from the core of *Tile D* is sent to the left (west) edge of *Tile E* during communication clock cycle one. During cycle two, connectivity is enabled to transfer data from *Tile E* to the West edge of *Tile F*. Finally, in cycle three the data is

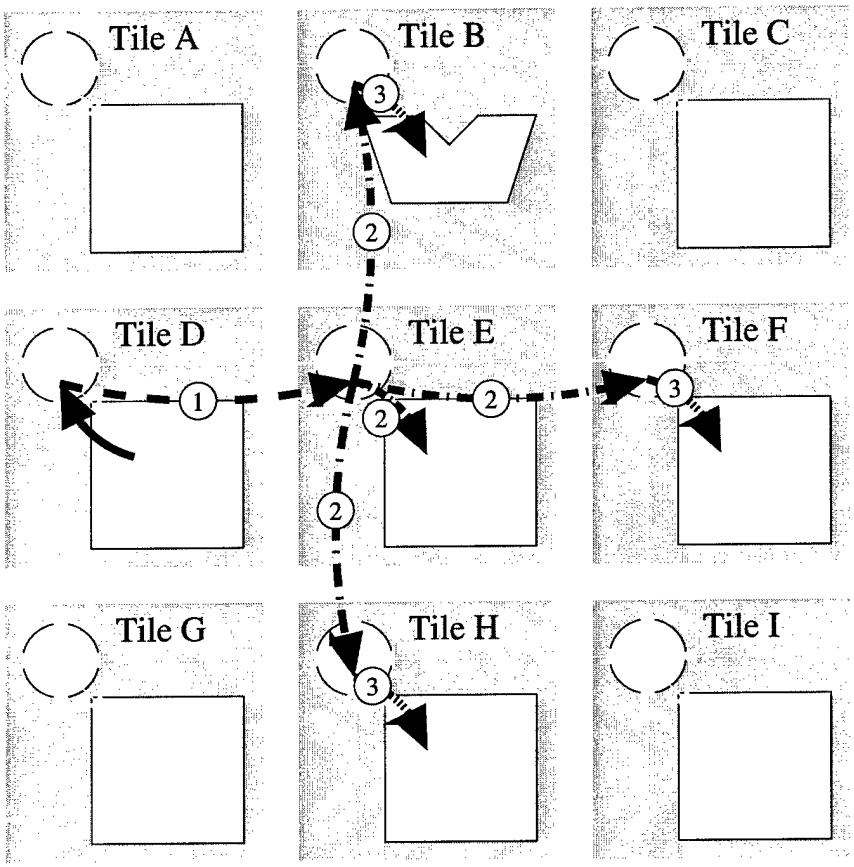


Figure 4.6. Multi-Sink Stream Communication

moved to its destination, the core of *Tile F*. During the same three cycles, data is transmitted from *Tile A* to *Tile E* in a pipelined fashion forming a second data stream. Notice that the data stream is pipelined and the physical link between *Tile D* and *Tile E* is shared between the two streams at different points in time. Stream sequences may be iterative. At the conclusion of the third cycle, the three-cycle sequence may re-start at cycle one for new pieces of data. The architecture supports streams with multiple destinations. At each interface the data from the input stream can be sent in up to four directions in the following clock cycle. Figure 4.6 shows a stream from *Tile D* to *Tiles B, E, F, and H*. In cycle two the data from *Tile D* is broadcast in all directions to meet the stream requirement.

The sequential and pipelined nature of data streams ensures data transfer with the following characteristics:

- All data in a stream follows the same source-destination path.
- All stream data is guaranteed to be transferred in order.
- In the absence of congestion, all stream data requires the same number of cycles to be transferred from source to destination.

Ideally, the exact time of all computation and communication for an aSoC application could be determined prior to run-time. In practice, variable data rates and run-time data dependencies make this approach infeasible. The data generation and consumption rates of cores in the same stream may differ significantly. If data is generated faster than it can be consumed, data congestion may occur in intermediate communication interfaces. Alternately, if data generation is slow compared to data consumption, data may be requested at the consuming core before it is available. To address each of these issues, data buffers are added to each interface. To facilitate communication between tiles, these data buffers are controlled with three flow-control bits. These bits identify the validity of the incoming data (*valid*), the success of the current transfer (*fail*), and whether the data had been previously sent (*resend*). Due to the pipelined nature of the architecture it is not possible to transfer data and check its success in the same clock cycle. As a result data, in the same stream cannot be transferred between cores on consecutive clock cycles. The resulting bandwidth penalty can be eliminated, however, by breaking high bandwidth communications into two separate streams at the cores and alternating them on the interconnect.

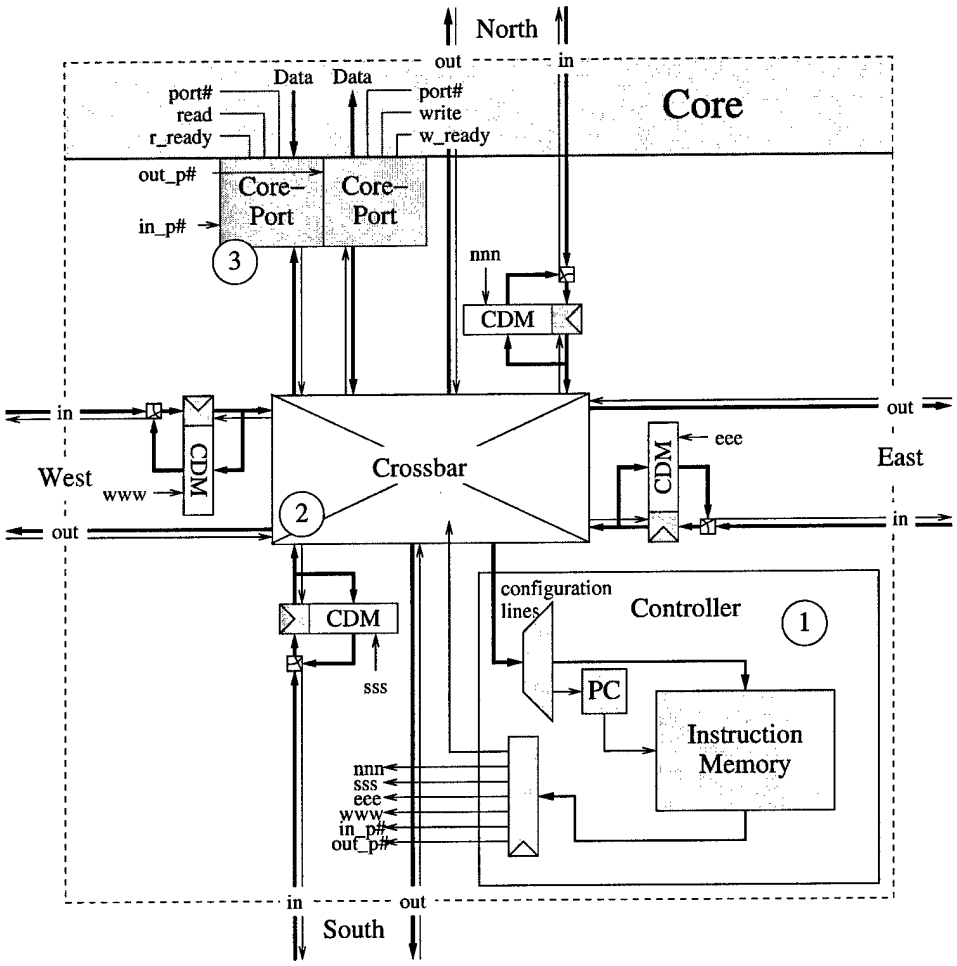


Figure 4.7. Core and Communication Interface

4.2.2 aSoC Architecture

To understand the implemented architecture, a single *communication interface* (CI), shown in Figure 4.7, is examined in detail. As seen in Figure 4.7, tile resources are partitioned into a distinct IP core and the communication interface, which coordinates communication with neighboring nodes. At a high level, the communication interface can be subdivided into three major components:

1. **Communication Controller and Instruction Memory (CCIM)**: Controls the connectivity of the CI to establish the time-multiplexed streams of communication.

2. **Data Flow**: The *crossbar* and *communication data memory* (CDM) provide in-order data transfer through the tile.

3. **Core-port**: Simple synchronizing interface to core.

In Figure 4.7, thin lines represent control signals and wide lines are actual data paths. To establish the desired streams, the *controller* in the CCIM reads the communication schedule from the *instruction memory* every cycle. Each schedule instruction contains the control bits necessary to set the connectivity of the *crossbar* and access the correct CDM buffers. The *crossbar* allows for data transfer from any input ($North$, $South$, $East$, $West$, and $Core - port$) to any output (five input directions and the configuration lines to the controller). If, due to congestion, it is not possible to complete a transfer on a specific clock cycle, data is stored in the CDM. The *core-ports* provide buffering for local transfers between the IP core and its communication interface. This structure serves as a synchronizing interface for the network and cores operating at different clock frequencies.

4.2.2.1 Communication Controller and Instruction Memory (CCIM)

A detailed view of the CCIM appears in Figure 4.8. The communication schedule is stored in a dual-ported SRAM-based *instruction memory*. Each schedule instruction is 40 bits and contains information sufficient to control the *crossbar*, *CDM*, and *core-ports*. Six bits, the *scheduled jump* (sj) and the *jump program counter value* (jpc), help control the *instruction memory* access pattern. The memory is capable of both read and write operations in the same cycle, with writes occurring in the positive clock phase and reads in the negative. The *instruction memory* can hold up to 32 instructions with the first two hard-wired for initialization tasks. This memory depth could be scaled to support schedules of various complexities.

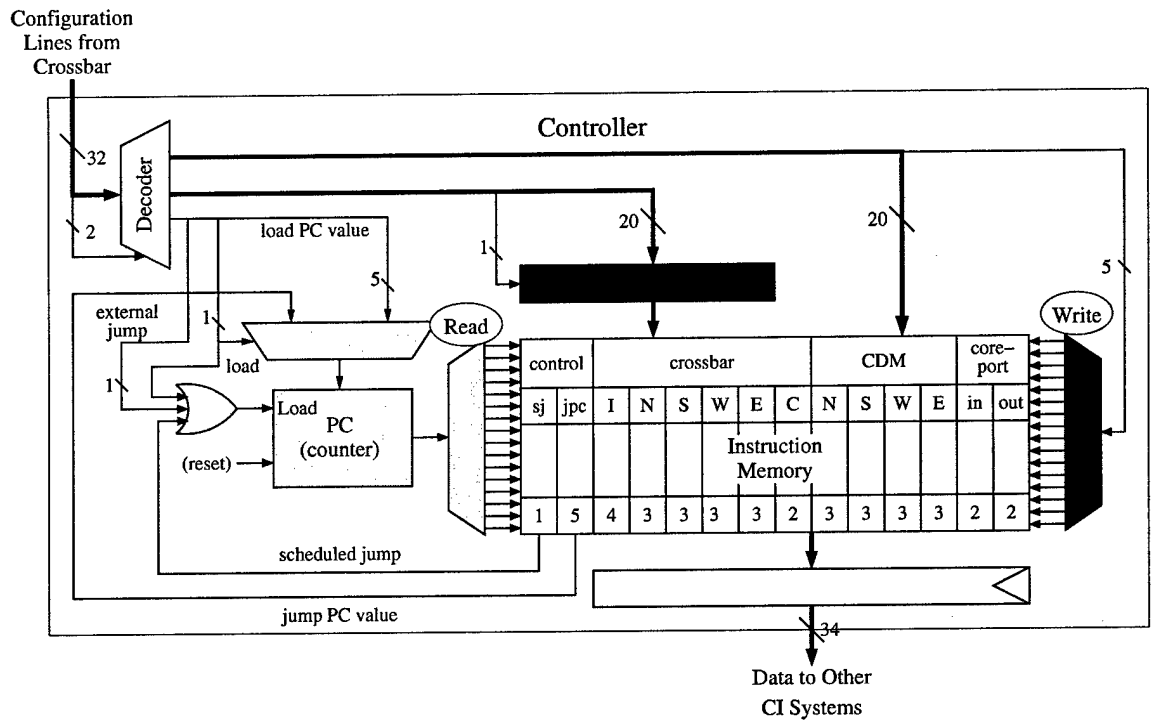


Figure 4.8. Detailed Communication Controller and Instruction Memory

The write circuitry, shown in dark gray in Figure 4.8, is used during initial configuration or at run-time for reconfiguration. Configuration commands come into the CCIM from the crossbar on 32 *configuration lines.* Two of these configuration bits are used to select the configuration operation through the input *decoder.* As the crossbar word size is smaller than that of the instruction memory, it takes two cycles to load a schedule instruction. One configuration operation loads a latch holding the first half of the new schedule instruction. The next configuration operation contains the second half of the schedule instruction and the write address.

The read address comes from a five-bit program counter (PC). Under normal operation the PC is incremented with each clock cycle to select the next consecutive schedule instruction. There are four ways of modifying the PC at run-time:

- A global reset signal, *(reset)*, can be applied at any time to set the PC to 0. This is critical during initialization.

Instruction #	Required Communications	sj	jpc
1			NA
2	**west to east**		NA
3	**west to core-port**	jump	Ins. 1

Table 4.1. Communication Schedule for Tile D

- A jump signal, *sj*, can be added to the schedule instruction to force the schedule to repeat. Table 4.1 illustrates an instruction memory sequence from the streaming data example of Figure 4.5. For *Tile E* the required stream communications are shown in boldface in instructions two and three of the communication schedule. As described in [34], communication resources are time-sliced on a per-communication cycle basis using space-time resource scheduling. If the streams are required for multiple words of data the schedule can be automatically repeated by setting the scheduled jump bit, *sj*, and pointing the jump PC value, *jpc*, to that of instruction one.

- To support dynamic communication patterns an *external jump* signal can be sent through the configuration lines. This is especially useful when an application requires data be processed in one of two different ways based on run-time information. Both schedules, provided they fit, can be loaded into the instruction memory and the *external* jump signal, *ej*, can be used to switch between them. Table 4.2 adds the schedule from the multi-sink example of Figure 4.6 to the original example in Table 4.1. An external jump signal received during instructions one, two, four, or five will push the PC to the complementary schedule. If care is taken in the development of the complementary schedules, this method of dynamic routing can be implemented on only the affected CIs without clearing the data in the network. Thus, it represents a single-cycle dynamic routing option.

Instruction #	Required Communications	sj	jpc
1			Ins. 5
2	**west to east**		Ins. 6
3	**west to core-port**	jump	Ins. 1
4			Ins. 2
5	**west to east & south & north & core-port**		Ins. 3
6		jump	Ins. 4

Table 4.2. Dynamic Routing by Run-Time Schedule Switching

- Finally, to support dynamic routing, a completely new schedule can be added to the instruction memory and started by loading its PC from the configuration lines. The *load* signal will force the PC to *load PC value* at the next cycle. This approach can be used for arbitrary dynamic routing but care must be taken not to eliminate active data streams.

4.2.2.2 Data Flow: Crossbar and Communication Data Memory

As a result of the unequal data consumption and generation possible in heterogeneous SoC, it may be necessary to buffer data at intermediate communication interfaces. In aSoC the communication data memory (CDM) provides one storage location for each stream that passes through a communication interface. To facilitate interface layout, this memory is physically distributed across the N, S, E, W sides of the CI. Figure 4.9 shows the data flow circuitry and the CDM buffers for the data arriving from the *West* direction, or *side*, of the CI. On a given communication clock cycle, if a data value cannot be transferred successfully, it is stored in the CDM. Three data bits for each *side* of the CI are required to assure proper flow of data between the tiles. To track real data in the network and at the destination core, a *valid* bit is attached to each word. A *fail_feedback* bit flows backwards to identify up-stream data congestion. This bit tells the up-stream CI to store the

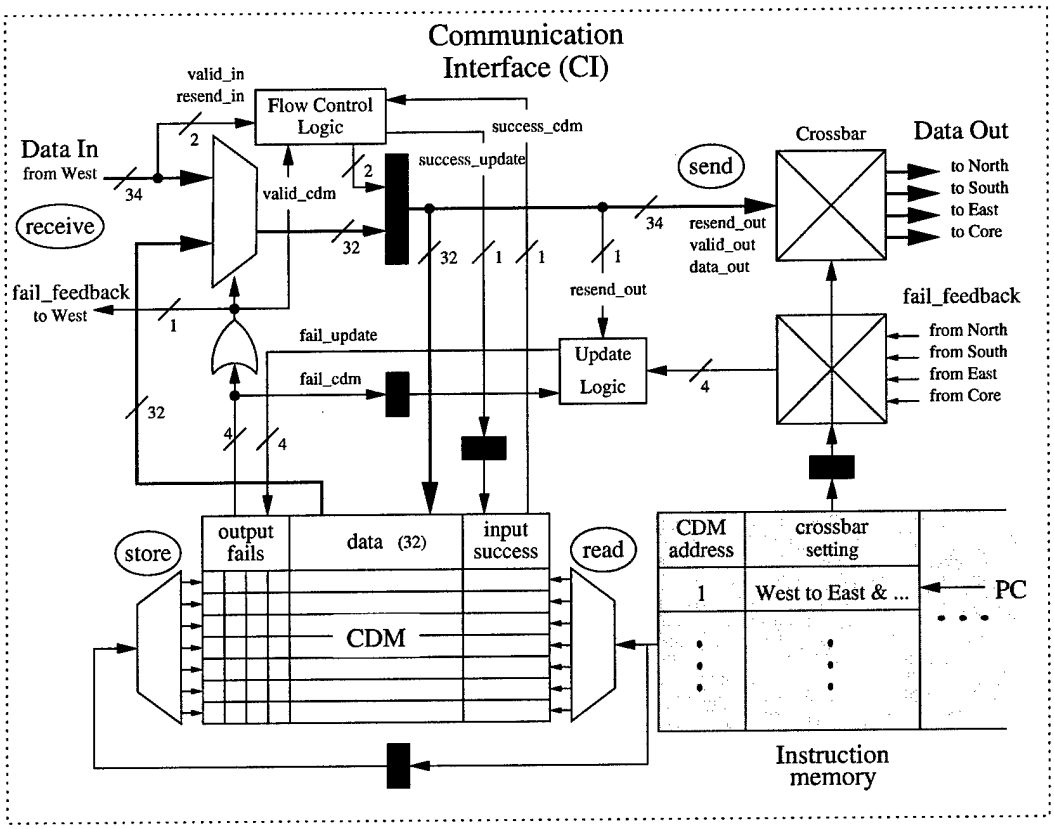


Figure 4.9. Flow Control Scheme

data for future transmission. Finally, an addition of this dissertation, the *resend* bit is attached to each data word to indicate if this data has been previously sent. This addition is required to support multi-sink communication streams, where congestion may occur unevenly among the branches. As such, proper data flow in the present architecture requires 34 forward-flowing bits and one back-flowing bit per transmission.

The present CDM supports the transmission of up to seven streams for each input *side* of the CI. This is scalable by increasing the address bit-width in the instruction memory. One CDM address per *side* is reserved as a no-operation (no-op) to reduce power consumption and ease compiler complexity. The CDM word holds the 32 data bits, a *fail* bit for each possible output destination, and receive-*success* bit to prevent duplication caused by re-sent data.

To fully understand the flow-control methodology it is helpful to step through a data transfer. A full data transfer requires two cycles, a read/receive cycle and a send/store cycle. In the read/receive cycle the *CDM address*, read from the *instruction memory*, is used to read the CDM buffers for each *side*. The CDM contains valid data awaiting transfer if any of the four *fail* bits are set. The logical **(or)** of these bits is used as the *fail_feedback* bit, which is fed back to the up-stream CI immediately. For the no-op *CDM address*, all the *fail* bits are 0. In the same cycle, data is received from the up-stream CI. A multiplexer at the receiving CI is used to select between the up-stream data and the data read from the CDM. To maintain in-order data transfer valid CDM data is always selected by the logical **(or)** of the *fail* bits. As such, the forward moving *valid_cdm* bit is logically equivalent to the back propagating *fail_feedback* bit. If the CDM data is valid, the received data will go unused and have to be retransmitted at the next instantiation of the data stream.

The *flow-control logic* in Figure 4.9 is responsible for generating the forward flow-control bits, *valid_out* and *resend_out*, and the *success_update* bit according to the logic in Table 4.3 and Equations 4.1, 4.2 and 4.3. In Table 4.3 x's are used to mark conditions, which are not permitted in the network. The input data must be valid if it is being re-sent, and the input data must be re-sent if the previous transfer was not successful. Behaviorally, the output valid bit, *valid_out*, is set whenever the CDM contains valid data, i.e. the *valid_cdm* bit goes high. The output data could also be valid if the input data is valid and it has not already been successfully passed to the present node. Given the do-not-care cases in Table 4.3, the *valid_out* logic is reduced to that in Equation 4.1. Behaviorally the data is being re-sent if it comes from the CDM and not the input side. The logic for this is trivial, as shown in 4.2. Finally, the receive phase is said to be "successful" if there is no valid data in the CDM or the input is empty. Additionally, previously successfully received

success_cdm	*valid_cdm*	*valid_in*	*resend_in*	*valid_out*	*resend_out*	*success_update*
0	0	0	0	x	x	x
0	0	0	1	x	x	x
0	0	1	0	x	x	x
0	0	1	1	**1**	0	**1**
0	1	0	0	x	x	x
0	1	0	1	x	x	x
0	1	1	0	x	x	x
0	1	1	1	**1**	**1**	0
1	0	0	0	0	0	**1**
1	0	0	1	x	x	x
1	0	1	0	**1**	0	**1**
1	0	1	1	0	0	**1**
1	1	0	0	**1**	**1**	**1**
1	1	0	1	x	x	x
1	1	1	0	**1**	**1**	0
1	1	1	1	**1**	**1**	**1**

Table 4.3. Logic Required for Forward Flow-Control Bits: *valid_out*, *resend_out*, and *success_update*

data is always consider a successful transfer. Using the do-not-care conditions the *success_update* bit is calculated with the logic of Equation 4.3.

$$valid\_out \;=\; valid\_cdm + \overline{success\_cdm} + (valid\_in \cdot \overline{resend\_in}) \tag{4.1}$$

$$resend\_out \;=\; valid\_cdm \tag{4.2}$$

$$success\_update \;=\; \overline{(valid\_cdm \cdot valid\_in \cdot \overline{resend\_in})} \cdot \overline{(\overline{success\_cdm} \cdot valid\_cdm)} \tag{4.3}$$

The dark gray latches in Figure 4.9 separate the read/receive cycle from the send/store cycle. In this cycle, the *crossbar* connectivity is established and the data is sent to a specified combination of the possible output directions. At the same time, the *success_update* bit and output data are written to the CDM using the previous *CDM address*. The *fail_feedback* bits from all possible data destinations are

resend_out	*valid_out*	*fail*	*fail_feedback_in*	*fail_update*
0	0	0	0	0
0	0	0	1	0
0	0	1	0	x
0	0	1	1	x
0	1	0	0	0
0	1	0	1	**1**
0	1	1	0	x
0	1	1	1	x
1	0	0	0	x
1	0	0	1	x
1	0	1	0	x
1	0	1	1	x
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	**1**

Table 4.4. Updating the Fail Bits

received and decoded in the *crossbar*. If a destination is selected, its *fail_feedback* is sent to the *update logic*. If not, a 0 is sent. The *update logic* is responsible for setting the *fail* bits in the CDM according to the logic in Table 4.4. Only one bit is shown in Table 4.4, as the logic is identical for each destination. The *fail* bits are set with the first feedback failure. The bits remain set as the data is re-sent until the feedback failure is cleared. Once cleared, the CDM *fail* bit can not be set again until the *resend_out* bit is cleared. This is represented in Equation 4.4.

$$fail\_update = \overline{\overline{fail\_feedback\_in} + (fail + (\overline{resend\_out + \overline{valid\_out}}))} \quad (4.4)$$

4.2.2.3 Core-ports: Connecting Cores to the Network

The aSoC *core-port* provides a synchronization and buffering resource between cores and the communication interface. Both core input and output ports contain dual-port memories. Each memory contains addressable storage locations for individual streams, allowing multiple input and output streams to be targeted to each

core. The structure of the ports allows other streams to continue transport if an individual stream is blocked.

The core-port architecture is designed to permit interfacing to a broad range of cores with a minimum of additional hardware, much like a bus interface. Both core-to-interface and interface-to-core transfer is performed using asynchronous handshaking to provide support for differing computation and communication clock rates. On the communication interface side of these ports, the handshaking is synchronous and uses a flow-control methodology compatible with that used by the data flow system discussed above. The interface to the core uses a simple two-step process, where the port sends a *ready* signal to the core and the core responds with a *read* or *write* operation.

Although the input and output *core-ports* are similar, it is more tractable to discuss them individually. Figure 4.10 shows the structure of the input *core-port.* The *core-port* memory and flow-control logic are synchronized to the rest of the CI. Simple control logic shown in Figure 4.10 prevents data duplication due to multi-sink data transfers. The flow-control bit, *fail_feedback*, is properly sent to the proper up-stream CDM through the crossbar. As the input data and flow control are only traveling within the local CI, the input *core-port* is not part of the system critical path and the entire write operation can occur in a half cycle.

The dark gray region at the bottom of the figure represents the core, which potentially operates with a different clock frequency and supply voltage than the rest of the chip. The core-to-interface handshaking protocol is shown in the timing diagram of Figure 4.11. In this protocol the core can monitor any of the three *ready* signals, one for each port buffer in the present design. Once the *ready* signal goes high the core can initiate a read by setting its *read* signal and sending the *port number.* After the data transfer is complete the *ready* signal will fall and the *data* will be valid.

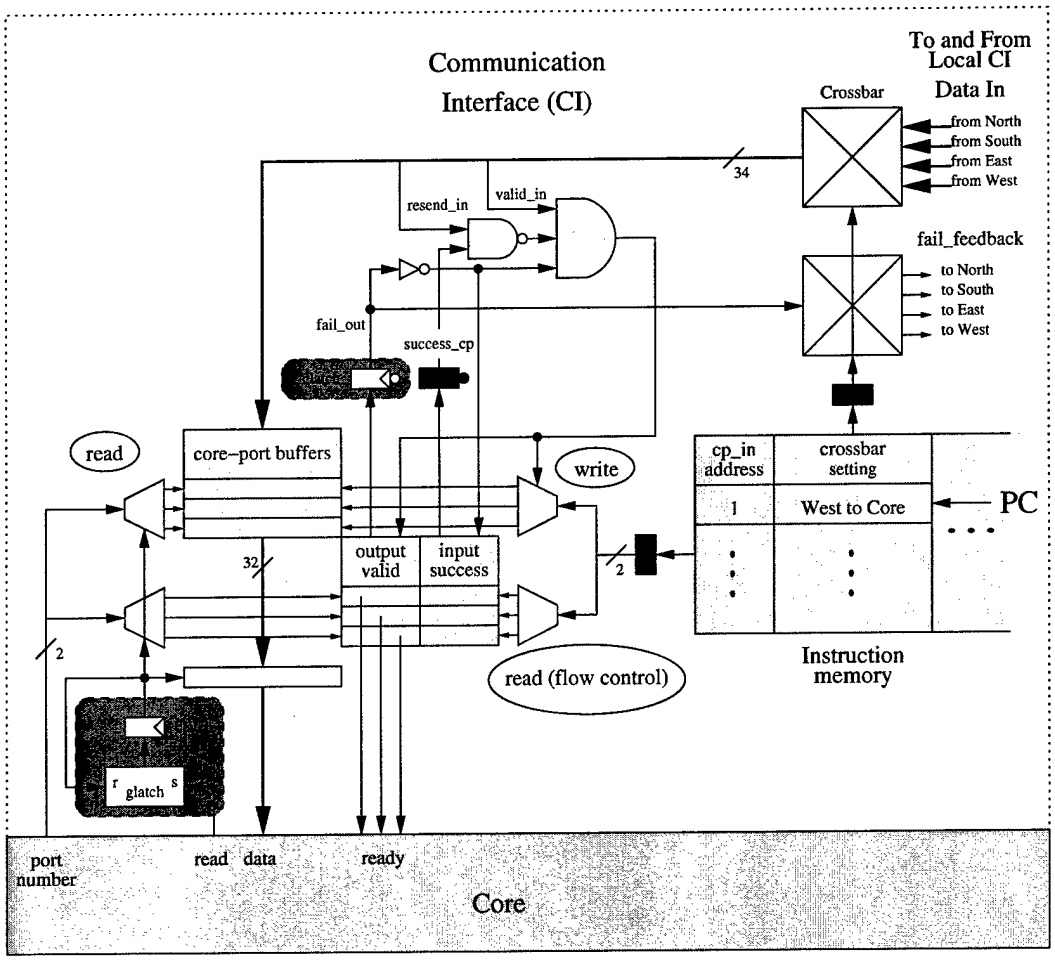


Figure 4.10. Input Core-Port Memory and Control Logic

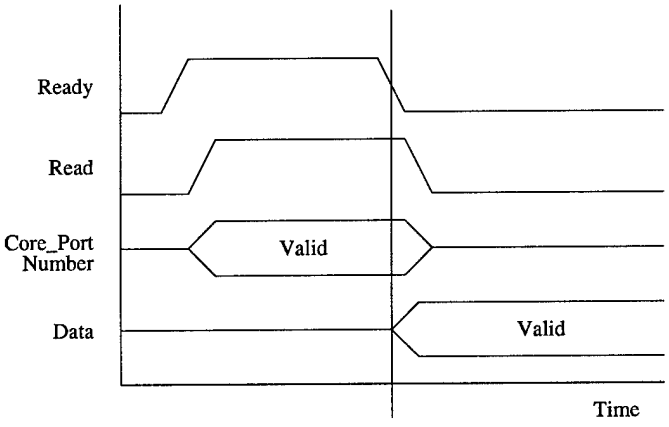


Figure 4.11. Input Core-Port Handshaking Timing Diagram

True synchronization between the two subsystems is not a trivial matter [160, 161, 162, 163]. Latch-based protocols, like the one proposed here, can experience problems when input signals crossing the asynchronous boundaries arrive too close to the latch timing signal. In these systems there is a finite probability that the latch ends up in a metastable state for a period of time. Error-free synchronization typically requires a *stoppable clock* approach [160]. For aSoC, however, it is possible to avoid the overhead of *stoppable clock* systems as the core clock is generated by division of the global clock used by the CI. As such, the skew between the two domains can be evaluated and even controlled with additional hardware.

Tuning the latches to reduce their susceptibility to the metastability problem creates a large range of error-free clock skews and reduces the criticality of the skew control. With this in mind, special latches, highlighted in dark gray in Figure 4.10, are developed for the critical *read* signal, which crosses the asynchronous boundary. The glitch-latch, *glatch*, is developed to track positive edges of signals. In this diagram it is set, *s*, on the positive edge of the core *read* signal and reset, *r*, after the read signal has been synchronized. This latch can not be set again until the core *read* signal goes low and comes high. As such, the read signal can stay high for many global clock cycles and still produce only one read.

The clear-latchs, *clatch*, are used for synchronization. These cross-coupled,inverter-based, edge-triggered latches are cleared during the second half of the global clock cycle. Clearing these latches every cycle makes metastability less of an issue. The *clatch* on the core side of the port forces the read operation to take place during the positive phase of the global clock. The *clatch* on the CI side forces the write operation to the negative phase of the global clock. This eliminates the potential for data collisions in the *core-port* buffers. Chapter 6 will show the results of these latches in operation.

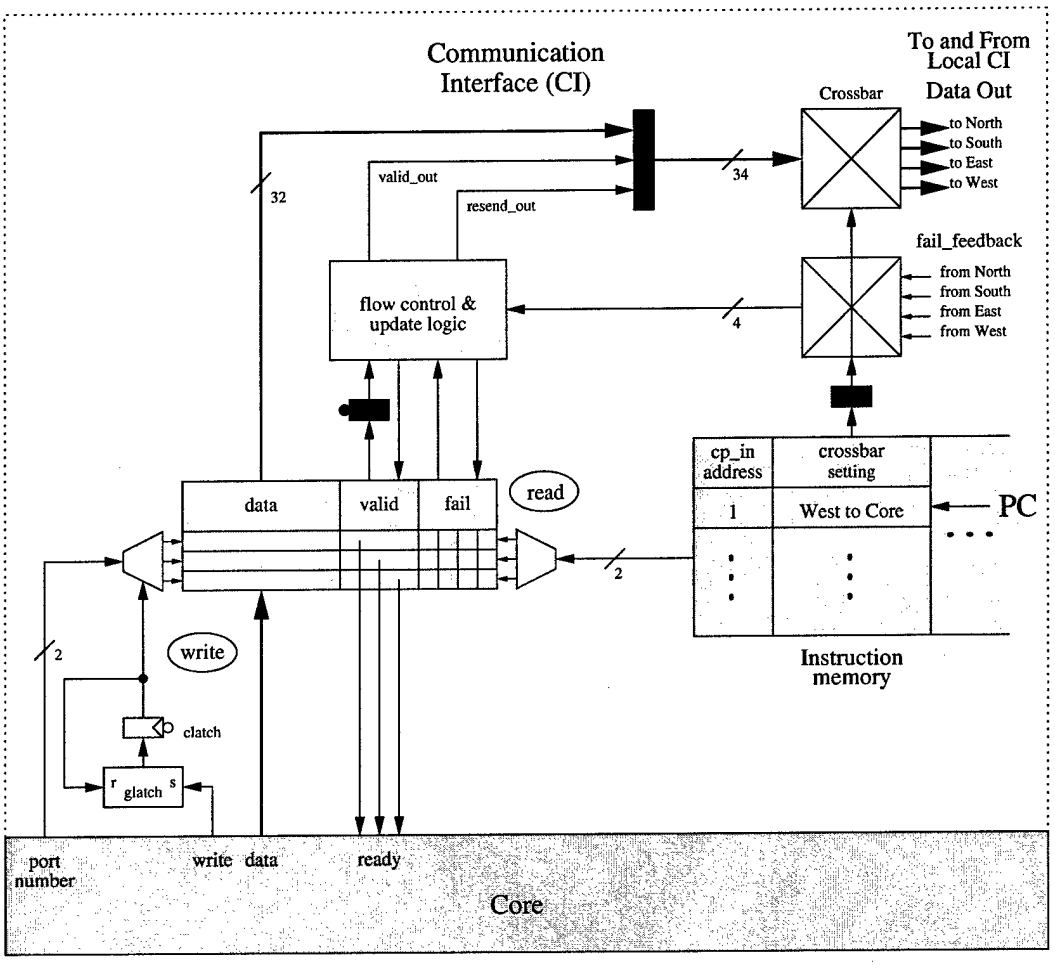


Figure 4.12. Output Core-Port Memory and Control Logic

The schematic for the output *core-port* is somewhat simpler, as shown in Figure 4.12. From the CI side of the port the data must be read from the memory in one cycle and sent through the crossbar to neighboring tiles in the next. The valid bit is latched on the negative edge of the first cycle. This prevents data collisions with core writes in the negative clock phase and allows time to calculate the *valid_out* and *resend_out* bits needed in the network. The *valid_out* bit is simply the *valid* bit stored in the *core-port* and *resend_out* is the logical **or** of the *fail* bits. The *fail* bits are updated using Equation 4.4 as done in the CDM. Care must be taken when updating the *valid* bit. To avoid destroying incoming data from the core, a tri-state

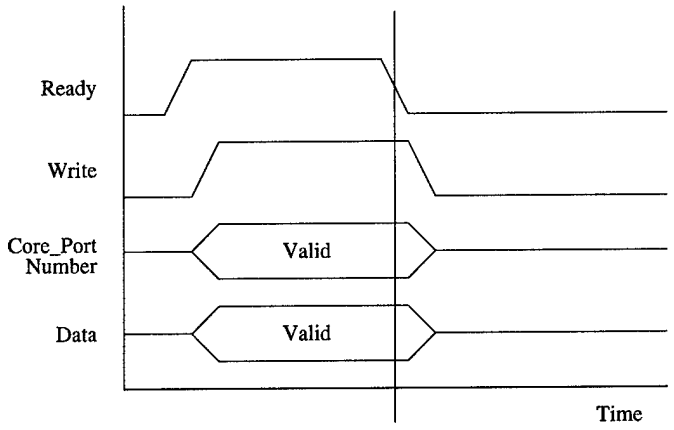


Figure 4.13. Output Core-Port Handshaking Timing Diagram

reset system is used to reset the *valid* bit low when all the *fail_update* bits return low and the valid bit was originally high.

The write from the *core-port* is very similar to the read as shown in the timing diagram of Figure 4.12. Again special latches are used to virtually eliminate the synchronization problem. In this case the clear-latch forces the write operation to occur in the negative phase of the global clock, which eliminates possible interference with the CI read.

Chapter 5

Using Dynamic Voltage Scaling to Make SoC Power-Aware

This chapter presents the parameterization of the aSoC interconnect and a hardware-speculative approach to voltage and frequency scaling.

5.1 Frequency and Voltage Scaling Approach

Dynamic parameterization for SoC is limited in this study to the parameters related to the system infrastructure. As the interconnect uses less than 2% of the system power, it is important to identify system conditions and infrastructure parameters that lead to a reduction in the power consumed by the cores [36]. Of the infrastructure parameters, the core supply voltage and local clock reference frequency most impact overall power. As stated in Chapter 2, these can be lowered if the core finishes its computations at a faster rate than the rest of the system. This creates the very simple parameter map shown in Figure 5.1 for each core. Cores with higher utilization are bottlenecks in the system and should be run at higher voltages and frequencies.

Measuring core utilization directly is not often possible as it would entail changing the core structure. Therefore, interconnect utilization is used as a metric to infer core utilization. For the statically scheduled aSoC network, unsuccessful data transfers between two cores are the result of the receiving core not being ready for data. In this case, either the receiving core is running too slowly or the sending core is running too quickly. Measuring the success rates of core input and output data can isolate the core, causing the data transfer failure. While this approach shows promise

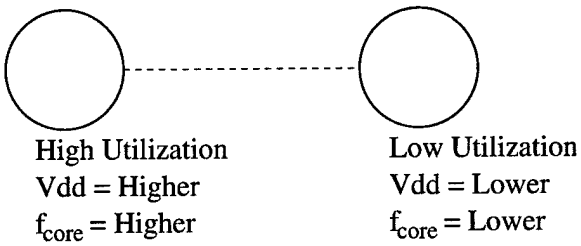


Figure 5.1. SoC Parameter Map Based on Core Utilization

as seen in Chapter 7, numerous pathological situations could arise when large numbers of cores communicate with each other. To handle these pathological cases, where cores interconnected in loops erroneously settle to the maximum or minimum frequencies, this system allows for user or software selection of core frequency. A more detailed study of the possible pathological cases is beyond the scope of this dissertation.

At each core, frequency and voltage can be automatically adjusted using a four-part system shown in Figure 5.2. After initial configuration through the interconnect *Local Configuration Lines*, a simple finite state machine successively increments or decrements clock frequency based on core utilization. The first subsystem, core *Utilization Measurement*, determines core utilization by accumulating read and write failures between the core and interconnect. This system evaluates both types of failures to increment or decrement the local core clock reference. The local *Clock Reference Selector* allows for the selection of 2^N discrete clock values over a wide range. To keep the system simple, local clock selection is limited to the frequencies immediately above or below the present clock rate. The N-bit clock selector register is connected to a look-up table (LUT) located in the *Voltage Selector* system. This LUT is loaded at compile time with predetermined data for each core. It is used to select the correct core voltage for proper operation at each possible frequency. Each of the four available supply voltages is connected to the core through a single PMOS pull-up device. Finally, as voltages and frequencies change, the proper timing

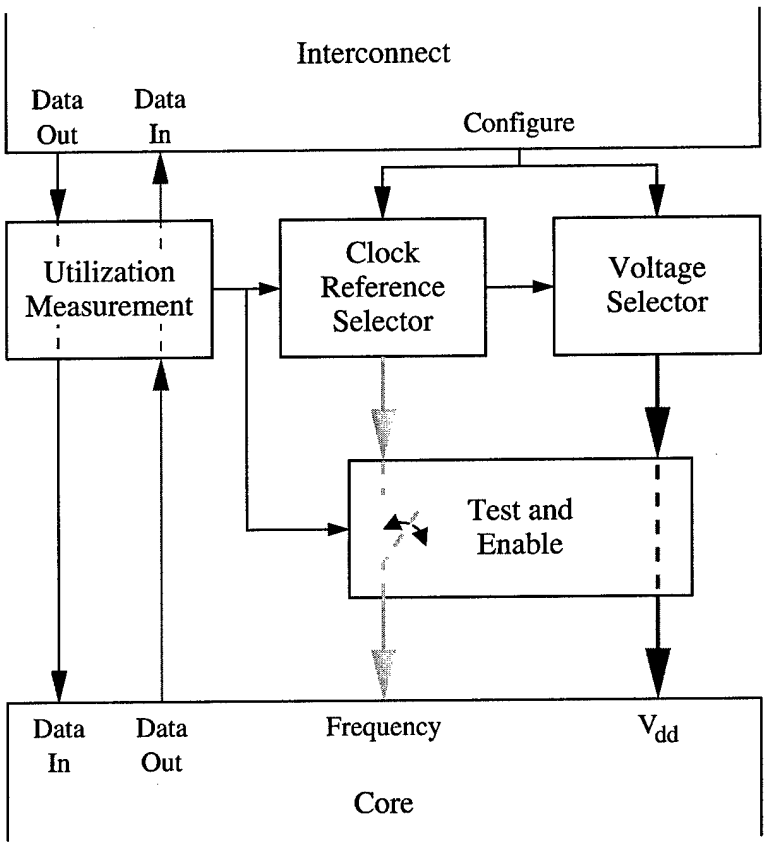


Figure 5.2. Dynamic Voltage Selection Block Diagram

in the core may not be preserved. To prevent timing violations in the core, the *Test and Enable* system uses a critical path model, specifically designed for the core, to evaluate core timing after a supply transition [6]. During this evaluation the clock reference to the core is disabled until simulated data can successfully pass through the critical path model.

The above system can potentially be used for any SoC topology and connectivity. The following subsections demonstrate the implementation for aSoC.

5.1.1 Core Utilization Measurement

Figure 5.3 shows a more detailed view of the core *Utilization Measurement* subsystem. For core *Utilization Measurement* an accumulator must be added to each *Core-Port*. Although the present aSoC implementation supports the use of

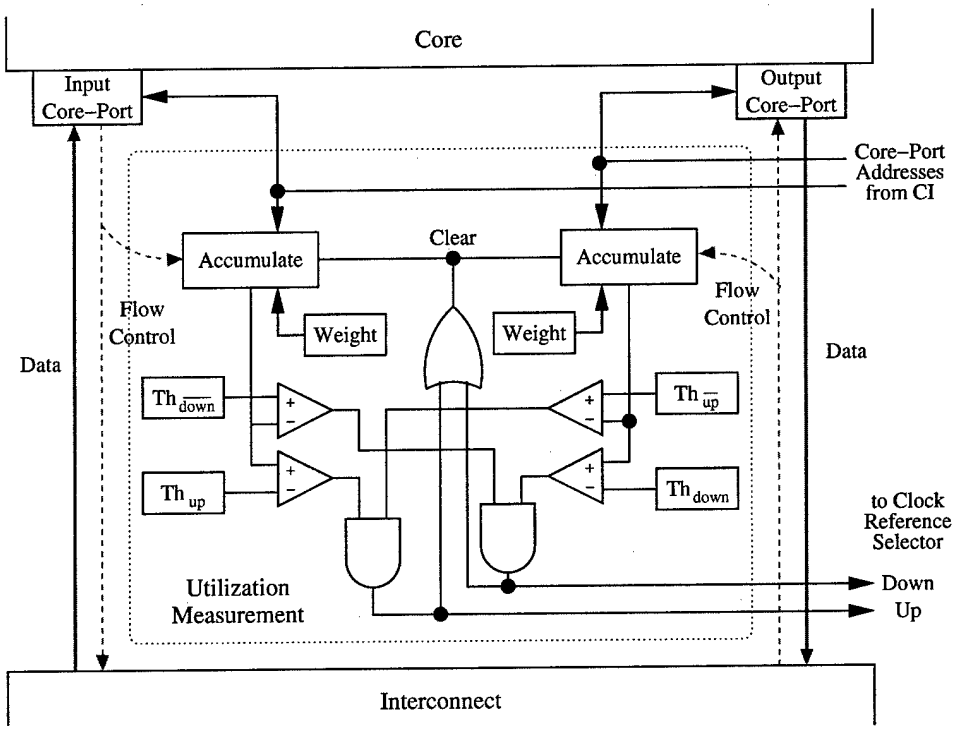


Figure 5.3. Core Utilization Measurement System

three input and three output *Core-Ports*, Figure 5.3 shows only one of each type. At present it is not known how many of the available *Core-Ports* should be used to monitor core usage.

The measurement system is enabled each time the first *Core-Port* address is requested by the communication interface. If the read or write operation is successful, the appropriate *accumulator* value is reduced. If not, the *accumulator* value is increased. The proposed system decreases the core frequency based on the accumulated *Output Core-Port* write failures. If the *Output Core-Port Accumulator* value increases above the threshold, Th_{down}, and the *Input Core-Port Accumulator* value has not reached the threshold, $Th_{\overline{down}}$, the *Down* signal is sent to the *Clock Reference Selector*. To increase the clock reference using the *Up* signal, the *Input Core-Port Accumulator* value must be above the threshold, Th_{up}, and the *Output Core-Port Accumulator* value must be below the threshold, $Th_{\overline{up}}$. After either the

Up or *Down* signals are sent, all the accumulators are cleared. At present the thresholds are hard-wired at design time based on the target application. Chapter 7 shows some of the issues in setting the threshold values and accumulator widths.

The key to this system is its reconfigurability. It may not be known at design time, the exact application is for each core in aSoC, or what type of data allocated for each core-port. Data streams critical to the application's throughput should be used to control core frequency, while less frequently used control data should be ignored. This is done by scheduling the critical stream in the first *Core-Port* address. To accomplish reconfigurability and accommodate streams of varying importance, the weight of success and failures for each accumulator can be set through the communication interface *Local Configuration Lines.* If the *Accumulator* failure weight is zero, the corresponding *Core-Port* can never cause frequency shifts. The relative success and failure rates can be used to compensate for streams that occur in the interconnect more often than they are needed. Low bandwidth streams may have to be scheduled more frequently than required, depending on the size of the communication schedule. At present the compiler and application mapping flow, AppMapper, does not allocate specific bandwidths to streams [37]. It simply finds the minimum number of instructions to assure all streams have connectivity [37]. As such, a stream may have multiple failures for each success and still meet system-wide throughput requirements.

In addition, each of the thresholds, Th_{up}, $Th_{\overline{up}}$, Th_{down}, and $Th_{\overline{down}}$, can be set depending on the core and application. These thresholds have the effect of low-pass filtering the failure data before adjusting core clock frequency. The bursty nature of cores, like the ME and DCT cores, may dictate differing levels of filtering to assure the failures are a result of stream blockages and not just the burstiness of the core.

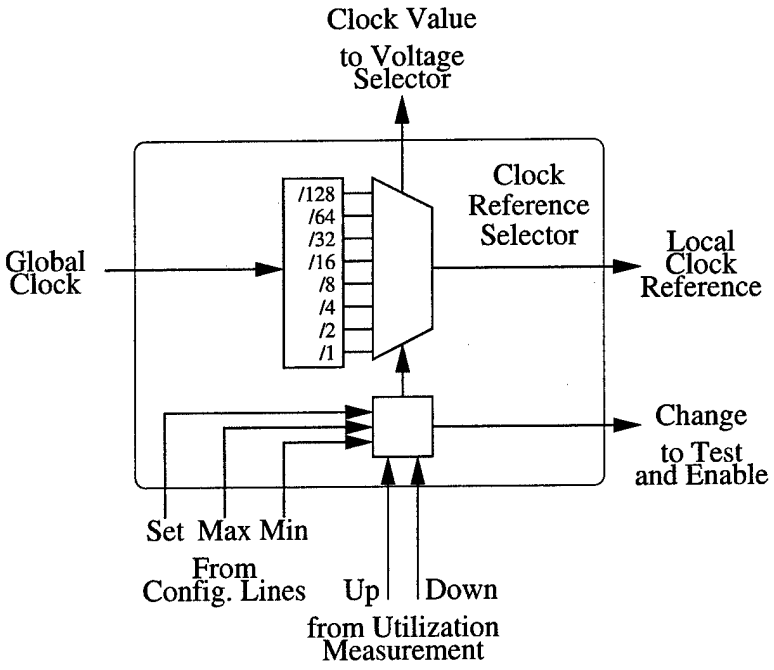


Figure 5.4. Clock Reference Selector Block Diagram

5.1.2 Clock Reference Selector

The local *Clock Reference Selector*, shown in Figure 5.4, selects between different frequencies. At present, eight frequencies are made available by successively dividing the high frequency global clock signal by multiples of two. A programmable three-bit up-down counter holds the state of the local clock frequency. During configuration the counter is set to some initial state based on static timing analysis. The counter is then incremented or decremented by the *up* or *down* signals from the core *Utilization Measurement* system. Any change in clock frequency is detected and sent forward to the *Test and Enable* system to temporarily disable the local clock. This is done to prevent data loss during clock and supply transitions. Maximum and minimum core frequencies are set at compile-time through the SoC configuration system. The system developed here allows the core to choose from eight frequencies. This number could be scaled up or down depending on the application. The state counter ignores any input which attempts to move the clock outside the allowed frequencies.

Search Window Size	Required Cycles
64 × 32	2000
16 × 16	512
8 × 8	192

Table 5.1. Motion Estimation: Number of Cycles vs. Search Window Sized

The state of the counter can be set at run-time using the configuration system. This allows for the use of different voltage scaling approaches. For example, in the *dynamically parameterized* motion estimation system described in Chapter 3, the core used previous motion vectors to control search window size. As a result, the core predicted how many calculations, and clock cycles, were required to process the upcoming macro-block, as shown in Table 5.1. If this information was made available to the core interface, it could be used to select a clock frequency appropriate for the next set of calculations. Likewise, compiler-driven voltage selection could be implemented in addition to this system by forcing the clock state up or down.

5.1.3 Core Voltage Selector

The core *Voltage Selector*, shown in Figure 5.5, uses a small LUT to choose the core supply voltage based on the selected clock frequency. The three-bit clock state is sent directly to the LUT, and the corresponding supply pull-up device is activated. The LUT is loaded at compile time based on a detailed analysis of core voltage and frequency characteristics. Each pull-up device, and associated multi-stage buffer, is designed to transition the supply quickly while providing the capability to deliver near-constant voltage at the worst-case current loads. Section 2.4 discusses the design issues for this system in some detail. In Chapter 2 it was shown that the voltage could be transitioned from one level to the next in relatively short times, $< 2ns$, even for large cores like the motion estimation core. A supply turn-off latch

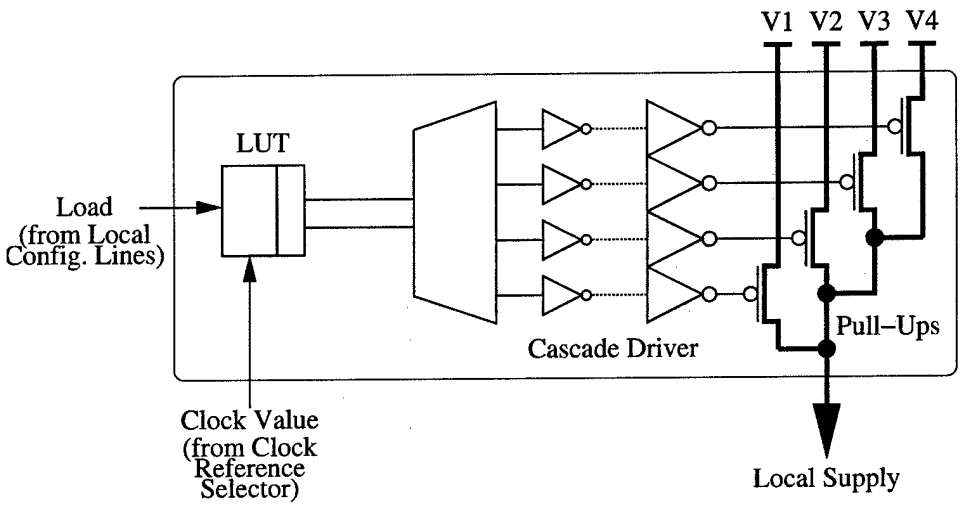


Figure 5.5. Core Voltage Selector Block Diagram

can be set through the aSoC configuration system. When enabled, all core supply pull-ups are disabled and the core can discharge to 0V.

5.1.4 Critical Path Test and Local Clock Enable

During each clock frequency transition the local clock is disabled in the *Test and Evaluation* system. After a short initial delay, test data is repeatedly sent through a set of dummy critical paths specifically tuned for the core they represent [6]. Figure 5.6 shows the simplest system. When the critical path in the core behaves nicely, it can be simulated by a simple inverter chain. For any selected voltage, the inverter chain will have a delay that is always at least as long as that for the true core critical path. At the end of the inverter chain, a latch is used to capture the delay information. Data sent through the simulated critical path is compared to data sent directly. If the two bits match, the local clock is enabled; otherwise, the core is frozen until the correct voltage has developed. As a result of this implementation, the local clock stays disabled for the entire voltage transition only when switching from low to high voltages.

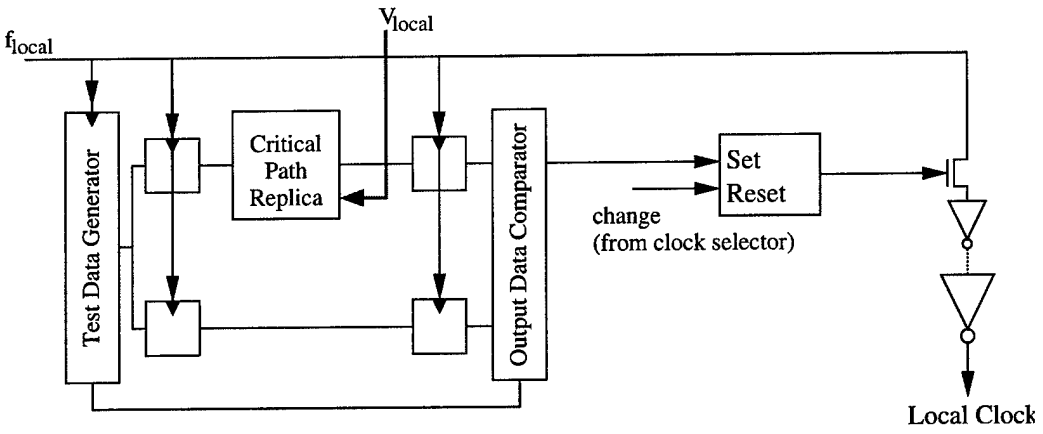


Figure 5.6. Simple Critical Path Test and Local Clock Enable System

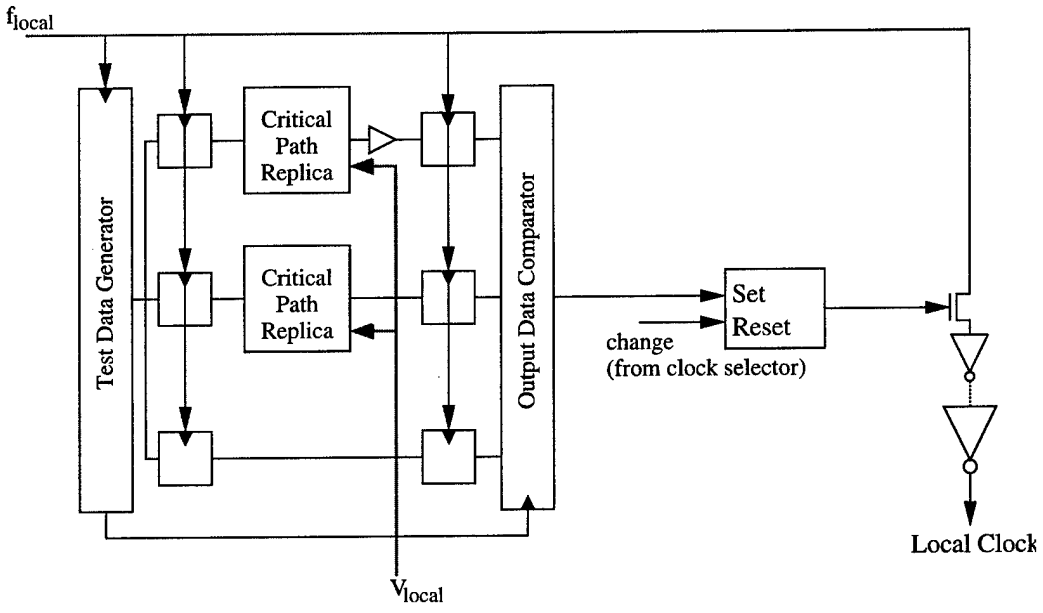


Figure 5.7. Race Safe Critical Path Check

The system is made only slightly more complex if there exists the potential for race conditions in the core. A third path is added to the critical path check, as shown in Figure 5.7. This third path represents an upper bound on the race delay. If data successfully makes it through this path in a single cycle, race conditions could exist in the core. The clock must stay off until data can no longer pass this path.

Finally, as discussed in Chapter 2, it is not possible to model the critical path of cores, which use pass-transistors extensively, with a simple inverter chain. The delay of the pass-transistor paths increases far more rapidly with voltage. In the best case, an inverter pass-transistor chain can be used to model the critical path. In the worst case, however, the critical path of the core may change. This would require separate critical path models depending on the voltage selected. This simplest solution is to develop a critical path model which bounds all the possible critical paths over all the possible voltages.

Chapter 6

SoC Design Methodology and Implementation of aSoC

To reduce power and increase performance, SoC endeavors to integrate entire system boards on a single chip. One of the main challenges is developing a system framework which can be implemented efficiently with potentially hundreds of heterogeneous cores. To make design time manageable, the SoC must support the re-use of intellectual property, and ease verification. In addition, the architecture must balance the need for performance while keeping power consumption down.

This chapter first investigates these key SoC challenges with respect to several proposed design approaches. To cope with these issues, the SoC physical design is highlighted as the foundation for realizable SoC architectures. ASoC infrastructure is developed to fabrication quality and evaluated to demonstrate the importance of physical design. This physical implementation approach is the most detailed proof-of-concept for any of the proposed large scale SoC architectures [30, 31, 32, 33, 35].

6.1 SoC Development Methodology

Throughout the last decade, reconfigurable architectures have been proposed as a way of coping with the dramatic increases in available circuit complexity predicted by Moore's Law. Specifically, these approaches have attempted to push performance and overcome increases in design time through either replication or reuse of resources. Systems like RAW [136], Colt [121], and MATRIX [117] trade hardware design for software mapping through the use of a homogeneous, FPGA-like, computing fabric. SoCs use standardized bus or interface systems, including

CoreConnect [154], AMBA [157], and VSIA [156] to allow the incorporation of pre-designed intellectual property (IP) cores.

Unfortunately, increasing global wire cost and power consumption have also become difficult problems in modern VLSI design. These additional problems threaten the scalability of the FPGA-like and bus-based SoC approaches. To reduce design time and the effects of increasing global wire cost while balancing performance and power consumption, many architects have turned to *tile-based* systems with *point-to-point mesh* networks [30, 31, 32, 33, 34, 35]. This architecture model reduces design time through the use of IP cores located in each tile. Uniform tile size, when coupled with point-to-point interconnection, limits wire length and equalizes wire delay. The mesh structure also enhances bandwidth, as communications from tile-to-tile can take various paths. As a result, this network architecture decreases global wire cost and improves performance over bus-based SoCs. Power consumption is going to be controlled through careful "Backbone" development [33] and trading quality of service (QoS) when necessary [146].

Gradually, as bus-based systems are replaced by these more complex routing networks, the term SoC is being replaced by the more descriptive term Network-on-Chip (NoC). This conceptual shift provides a formal abstraction hierarchy to on-chip network design through the application of the International Standard Organization's open system interconnect (ISO/OSI) network layer model, shown in Figure 6.1 [146]. In order to achieve this abstraction, a mapping of between-network and VLSI-architecture terminology is accomplished by L. Benini and G. De Micheli [146]. At the top of the network layer model, the *Session*, *Presentation* and *Application* correspond to application and operating system software. The actual NoC architecture consists of the *Data Link*, *Network* and *Transport* layers. And, finally, at the base of this abstraction, the physical layer corresponds to wires, which carry global communications. This linkage between the layer model and VLSI architecture

terminology allows for the application of network fundamentals to on-chip systems. It creates a network-centric development approach where network functionality and optimization can occur with a level of independence from global wire design. Of course, physical design must be considered, as the end performance of the network is tightly connected to the performance of the wires.

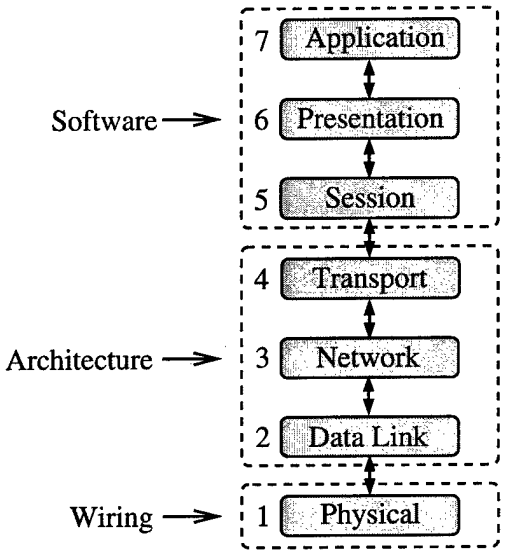


Figure 6.1. ISO/OSI Reference Model

In contrast to the network-centric approach, S. Kumar et al. proposes a three-phase development model, which starts with the construction of system infrastructure [33]. This approach, shown in Table 6.1 [33], begins by looking at the physical platform. The "Backbone Development" strives to account for the two-dimensional nature of silicon resources even in the selection of the network. As such, this phase includes both the physical and architectural layers of the network layer model. "Platform Development" is more closely linked to the architectural layer but still contains physical characteristics in the selection of tile size. Finally, "Application Development" clearly corresponds to the software layer. This approach represents a bottom-up design flow where interconnect evaluation is required from the beginning.

Table 6.1. Bottom-Up Design Methodology [33]

Phase	Tasks
Backbone Development	Region types Communication channels and switches Network interface of resources Communication protocols (specification)
Platform Development	Region scaling Resource design (units, interconnections) Dedicated hardware blocks System level control (implementation of communication, diagnostics, monitoring)
Application Development	Resource level control (OS) Functionality of resources (SW, configurable HW) Control of network Functionality of network

The reality of NoC design is most likely a combination of the network-centric [146] and bottom-up [33] approaches. While the network-centric network layer map perspective allows for network optimization, the S. Kumar et al. approach places emphasis on the physical implementation. Both aspects are critical to the realization of high-performance on-chip networks. To further bridge the gap between these two design methodologies, a new SoC/NoC development flow is proposed. Table 6.2 shows this development flow in three phases.

The development methodology in Table 6.2 combines the network model vertically, and the bottom-up approach horizontally. Phase one of this approach is an expanded version of S. Kumar's "Backbone" development [33]. Notice that it crosses all layers in the network map. Unlike the original "Backbone," software is included, as an architecture is useless without a way to use and test it. As such it will be called *infrastructure development.*

In phase one, most of the tasks are highly dependent on the network decisions. The most important relationship, from the perspective of this document, is co-dependency of the network decisions and physical floorplan. Ideally, network selection based on system performance should be the foundation for the other tasks.

Table 6.2. Full Development Methodology

Network Layer	Phase 1 **Infrastructure** **"Backbone"**	Phase 2 **Platform &** **Application**	Phase 3 **Fabrication &** **Test**
Physical	1. Floorplanning - Clocking scheme - Power distribution - Interconnect wires - Test structures 2. Resources - Network routers or switches - Cores - Interface	1. Select floorplan - Tile-size - Signaling technique 2. New core synthesis	Fabrication Test
Architecture	1. Network - Connectivity - Routers or switches 2. Cores - Interfaces - Dynamic Parameterization	1. Select network 2. Select cores 3. Develop new cores - Interface & parameterize	
Software	1. Network control (OS) 2. Simulator 3. Application mapping tools	1. Application software 2. Application mapping 3. Evaluation	
Result	Library of fully developed and tested system resources for plug-and-play design	Functionally correct design implementation optimized by selected SoC structure	Fully functional integrated circuit

Realistically, the physical limitations of the two-dimensional medium restrict the network design space. Finding an optimal network strategy is worthless if the cost of its implementation is unmanageable.

The concept for this flow is based on development methodologies common in today's markets. FPGAs companies like Xilinx [72] and Altera [71] already support the design methodology in Table 6.2 with phases one and three completed by the vendor. IBM also supports it with its CoreConnect product [154]. In these cases the vendor develops the architecture, software, and possibly even the hardware. A SoC developer would then choose from the pre-designed components provided by the vendor. The pre-designed tools and simulators help developers construct and evaluate their designs virtually. After this is complete, the design is functionally correct and can be sent for fabrication. Although all three phases are critical, phase three is the job of a fabrication company, and, as such, is beyond the scope of this discussion.

The rest of this chapter focuses on the physical development of SoC infrastructure shown in the top left corner of Table 6.2. Chapters 3 and 5 describe the application of *dynamic parameterization.* J. Liang's work [37] describes the development of other infrastructure components.

6.2 SoC Physical Infrastructure Development

SoC physical infrastructure includes issues ranging from floorplanning to power distribution to custom circuit design. The goal is to invest development energy in a design that can be widely reused across many SoC variants.

6.2.1 aSoC Floorplanning

In order to create a SoC physical infrastructure, silicon resources must be partitioned to perform specific tasks or house intellectual property subsystems. This process, *technology resource partitioning*, creates a system floorplan before any subsystems are developed. For this discussion, a MOSIS version of the TSMC $180nm$ process is used to demonstrate the infrastructure development [164]. As a result, seven process layers must be partitioned for the different functions within the SoC. The lower four layers, silicon active area–including the transistors and substrate–and three local metal layer are divided in the aSoC floorplan as shown in Figure 6.2.

Although Figure 6.2 shows a SoC with tiles that have 33kλ long sides (2.97mm in the .18μ technology), tile size can vary from design to design. The area overhead is equal to the fixed areas of the Communication Interface (CI) and Voltage Selection Systems added to the area, which may be wasted by the rigid floorplan of the SoC. The values for overhead areas are measured from layout models of these components. As such, the selection of tile size presents an interesting trade-off. Once the data width and instruction memory size of the CI are fixed, increasing the tile size decreases the infrastructure overhead. Conversely, as no active area is allocated for

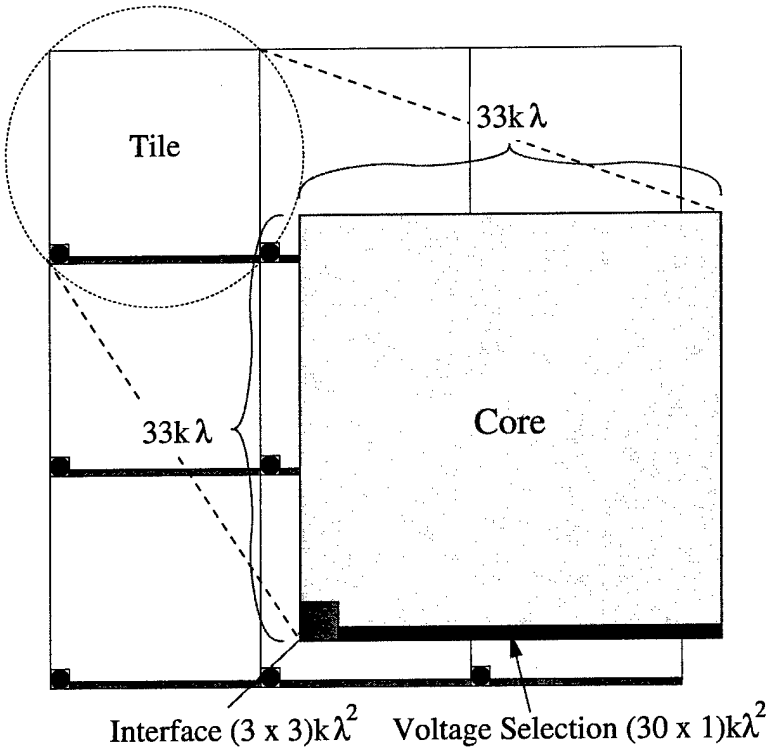


Figure 6.2. Active Area Floorplan

repeaters along the interconnect paths, the critical path is increased quadratically with tile size. In aSoC, the rigid floorplan of Figure 6.2 allows for optimal repeater insertion at the cost of having to design cores around the repeaters; but for simplicity, repeaters are not used in this project. Equation 6.1 describes the efficiency of active area usage.

$$\eta_{active\ area} = \frac{A_{tile} - A_{CI} - A_{Volt\ Select} - A_{waste}}{A_{tile}} \tag{6.1}$$

Wasted area is potentially one of the pitfalls of the rigid system floorplan. ASoC is designed to support heterogeneous cores with varying sizes. When tiles are made large, smaller cores use only a fraction of the desired area, thus increasing the system overhead. To reduce this overhead, smaller tiles could be used with core sizing options as shown in Figure 6.3.

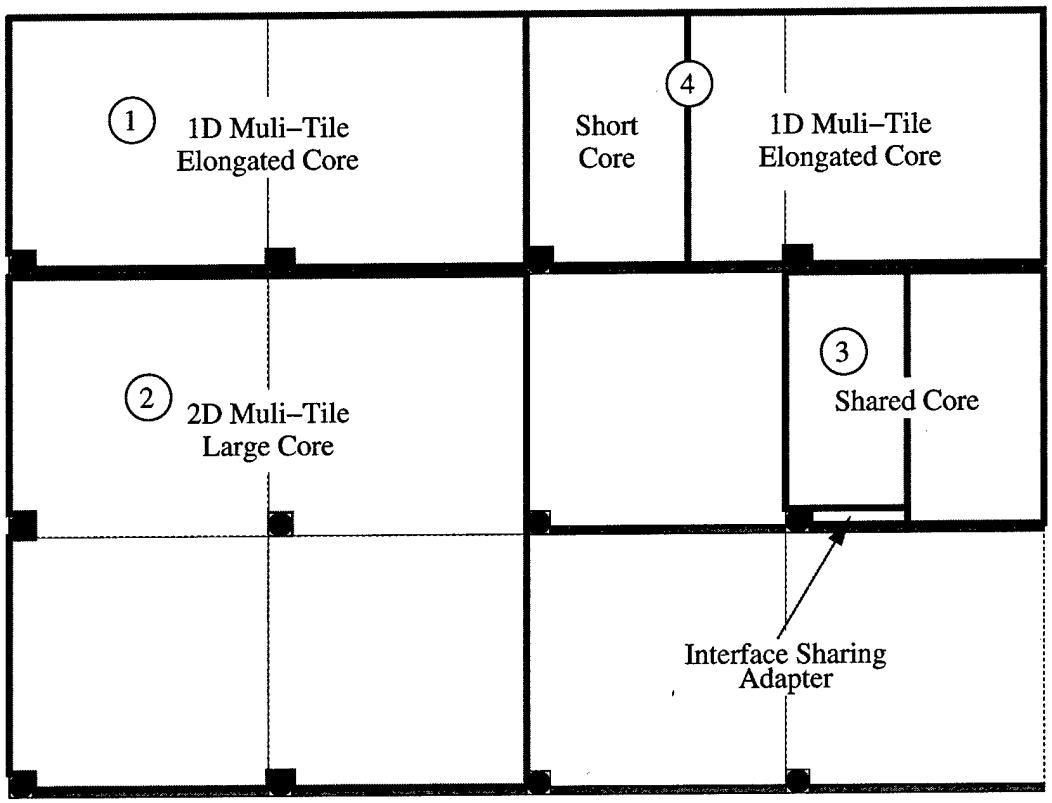


Figure 6.3. Core Placement Options

1. Cores could be elongated in one dimension to take any number of tiles in a row. This makes core placement slightly more difficult, but has the added benefit of providing larger cores with multiple communication interfaces.

2. Cores could be expanded in two dimensions providing space can be allocated for the (CI). This makes using existing core layouts more difficult.

3. Smaller cores, which are often used together, can be combined. The issue here is these cores now share one CI, clock reference and supply voltage. As a result, special interface sharing addapters will need to be constructed to time multiplex access to the core-ports.

4. Larger cores can use space left over by smaller cores. This represents a more difficult placement problem.

Core	Source	Core Size (λ^2)	Tile Size (λ^2)	Number of Tiles	Area Usage
Motion Estimation	P. Jain [23]	1.87G	1.09G ($33k \times 33k$)	2 (elongated)	86%
			.784G ($28k \times 28k$)	3 (elongated)	80%
			.529G ($23k \times 23k$)	4 (elongated)	88%
DCT	S. Venkatraman [24]	.068G	1.09G ($33k \times 33k$)	1	6%
			.784G ($28k \times 28k$)	1	9%
			.529G ($23k \times 23k$)	1	13%
ARM1136JF-S	ARM [81]	1.92G	1.09G ($33k \times 33k$)	2 (elongated)	88%
			.784G ($28k \times 28k$)	3 (elongated)	82%
			.529G ($23k \times 23k$)	4 (elongated)	91%
ARM1136J-S	ARM [81]	1.59G	1.09G ($33k \times 33k$)	2 (elongated)	73%
			.784G ($28k \times 28k$)	3 (elongated)	67%
			.529G ($23k \times 23k$)	4 (elongated)	75%
Single Port SRAM	Cell-Based Estimates	1560/cell	1.09G ($33k \times 33k$)	70KB/Tile	96.4%
			.784G ($28k \times 28k$)	50KB/Tile	95.6%
			.529G ($23k \times 23k$)	33KB/Tile	94.5%
FPGA	Macrocell Estimates 4-LUT CLB	7.18M/cell	1.09G ($33k \times 33k$)	121 CLB/Tile	80%
			.784G ($28k \times 28k$)	81 CLB/Tile	74%
			.529G ($23k \times 23k$)	49 CLB/Tile	66.5%

Table 6.3. Core Sizes and Active Area Usage

To ease placement-related issues, it is best to set tile size to best fit the bulk of the cores available. Table 6.3 shows a short list of cores and area utilization for various aSoC tile sizes. Only one-dimensional elongation of cores is considered. From this table, it can be seen that there is potentially a large variation in core size. As a result, there is great variation in the utilization of active area.

6.2.2 Clock, Supply, Global Interconnect, and Chip I/O

Up to now, a great deal has been done to establish the uniformity of the aSoC physical infrastructure. The primary justification comes through the discussion of the global interconnect, clocking and power supply delivery. Figure 6.4 shows the allocation of the highest three metal layers in the process. The highest metal layer runs horizontally and carries the global supply voltages, the horizontal running network and clock interconnects. To make the figure readable, only one set of horizontal voltage supply stripes is included. Replicas of these cover the entire die. The next layer runs vertically and contains global interconnect, clock tree wires and global power supply stripes. The last layer contains the local power supply network.

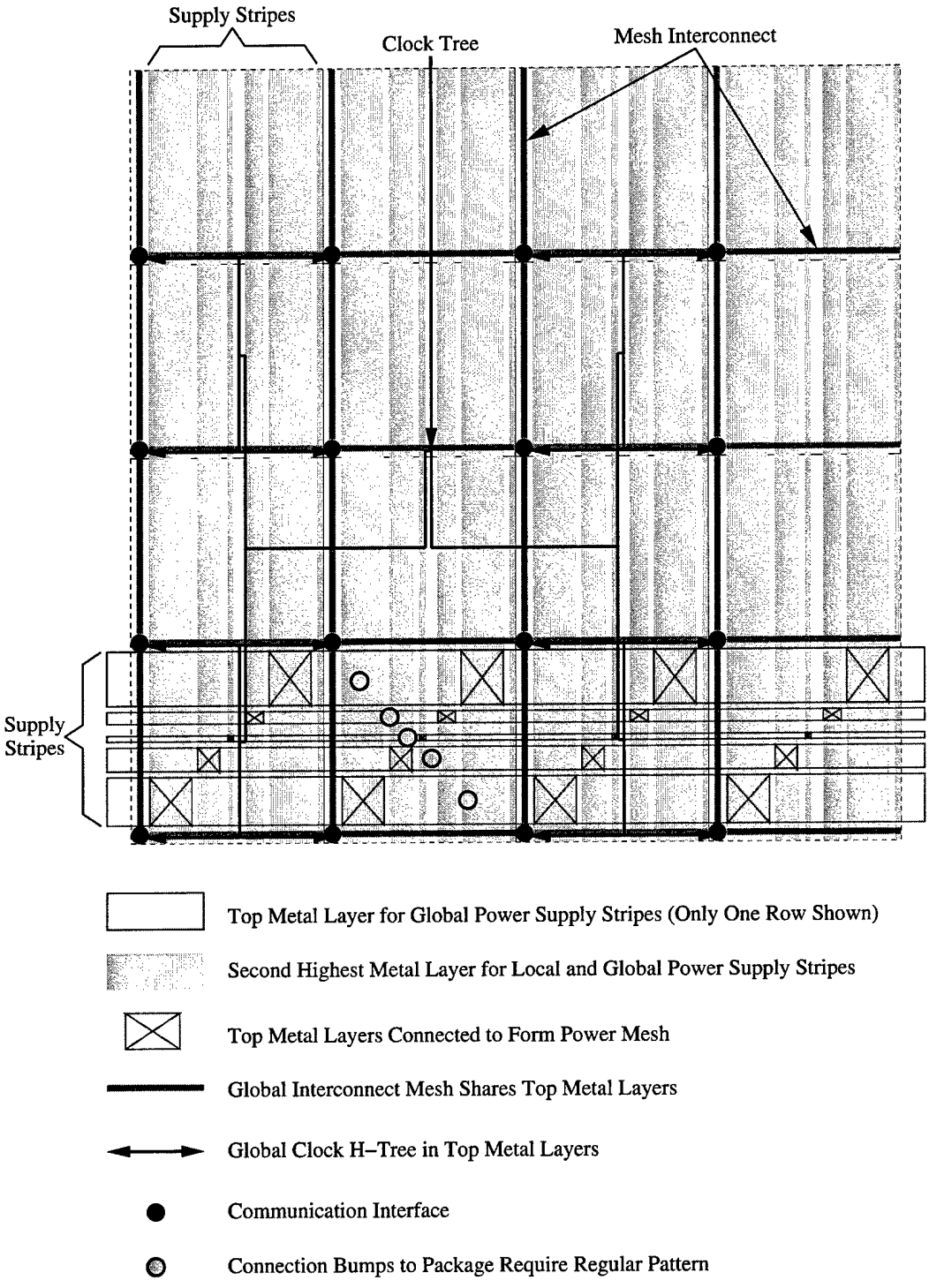


Figure 6.4. Global Metal Layer Allocations

The results for interconnect, clock distribution and power distribution represent simple case studies in the design space aimed at proving the feasibility of the aSoC physical infrastructure. They are not highly optimized, and all possible worst-case analyses are not completed. With this understanding, the examples clearly indicate the potential for successful physical implementation of aSoC and give a rough measure of the specifications possible.

6.2.2.1 Interconnect

The modularity and uniformity of the aSoC architecture allows for direct, point-to-point connection of wires in the global interconnect mesh, shown in Figure 6.5. This removes all routing issues and actually allows the mesh interconnect to reside in the same metal layers as clock and power. The interconnect length, L = Tile size - CI size, is the same for all tiles and directions in the architecture. This makes all routing delays the same and known before application mapping. As such, the delay can accurately be used in the evaluation of application mapping and core placement.

Table 6.4 shows the wire delays for several technologies and tile sizes. Unrepeated RLC wires are used in a 5-pi model to collect data. Notice that the wire delays allow for very fast network cycle times, even with the unrepeated lines. Additionally, performance could be improved if more advanced signaling techniques are used [75].

Tile Size	**Technology**			
	70nm	100nm	130nm	180nm
$18k \times 18k$	5.39	8.89	14.1	19.5
$23k \times 23k$	8.81	16.2	23.0	31.9
$28k \times 28k$	13.1	23.9	34.0	47.2
$33k \times 33k$	18.1	33.2	47.3	65.6
$38k \times 38k$	24.0	44.1	62.7	87.0

Table 6.4. Interconnect Delay (ps) with Non-Repeated Point-to-Point Signaling

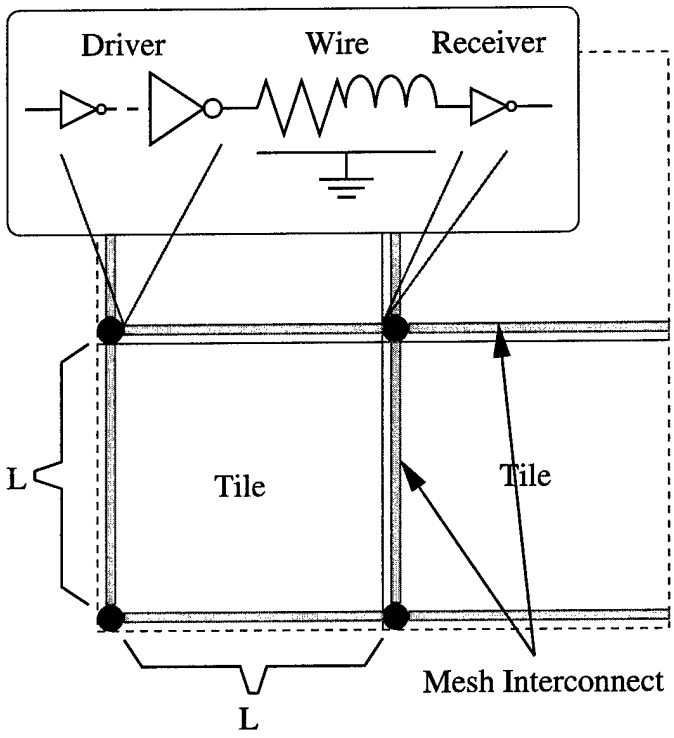


Figure 6.5. Interconnect Mesh Allocation

6.2.2.2 Clocking

Global clocking may be one of the most difficult problems in deep sub-micron design. Clock distribution can dominate system power dissipation and clock skew can seriously limit the system clock rate [70], which in this case directly relates to interconnect bandwidth. The uniformity of the aSoC network specifically addresses these issues and extends the usefulness of globally clocked systems. First, this uniformity allows for the design of a very precise H-Tree structure, as shown in Figure 6.6. Triangles in this figure indicate clock tree buffer placement. These buffers are placed precisely in the overhead region of the tile so that they do not interfere with the independent development of cores. This creates non-delay optimal buffer placements, with three different types of wire segments, as shown in Figure 6.6. Delay, however, is not a critical concern so long as the skew between clock terminals can be controlled.

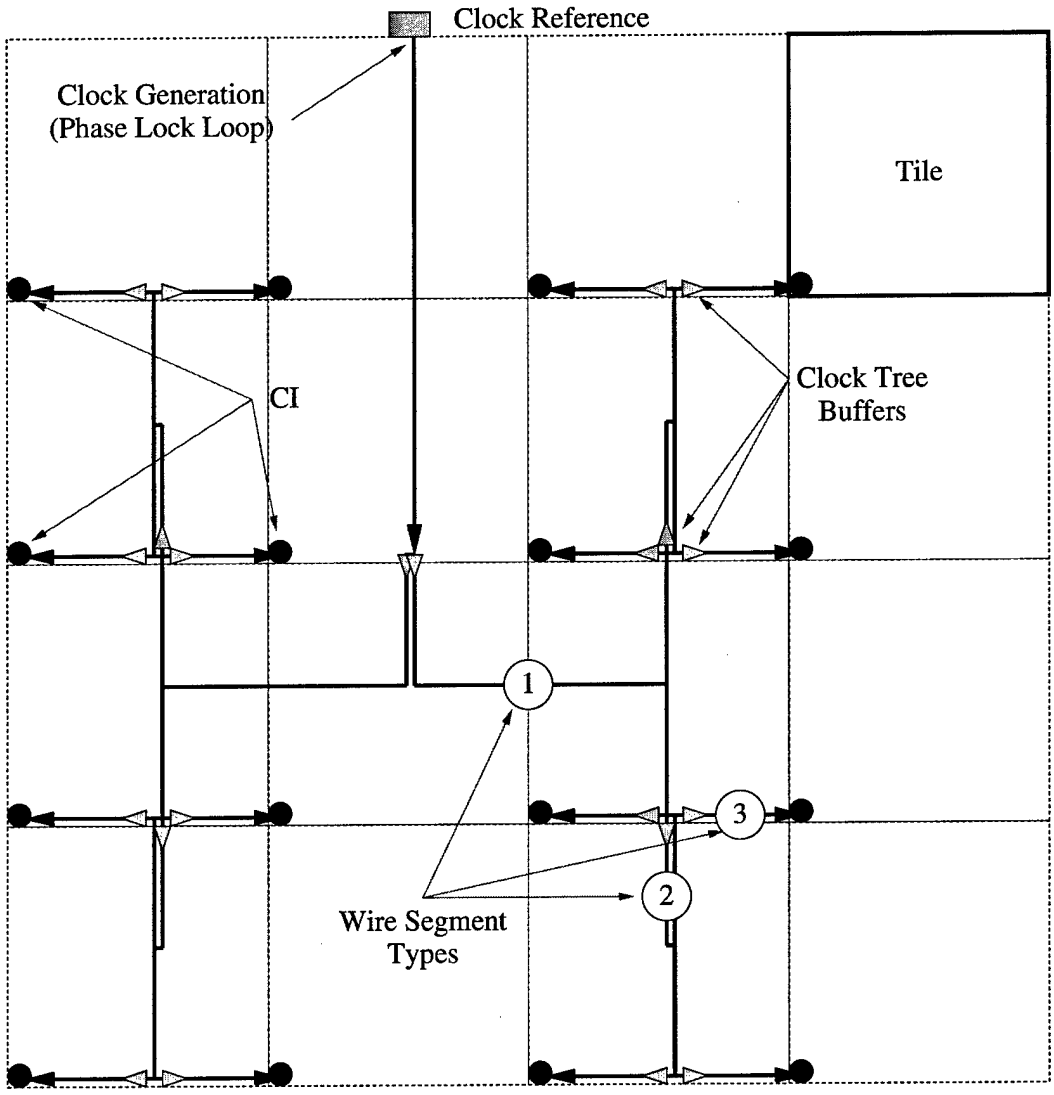


Figure 6.6. Ideal H-tree Clock Distribution Network

In order to demonstrate the benefits of the uniform floorplan, a simple clock structure is designed and evaluated with HSPICE. For simplicity, tile size is set to 30kλ on a side resulting in the *chip area* shown in Table 6.5. In this table, 16, 64, and 256-tile systems are considered. The H-Tree is modeled using 5-pi RLC structures for every half tile of wire, with buffer placement as shown in Figure 6.6. Capacitance, inductance and resistance for each wire segment are found using Berkeley predictive technology models (BPTM) [68]. The total capacitance of the

clock tree is shown as it relates to the power. The clock frequency is chosen such that it is consistent with those predicted by the International Technology Road Map for Semiconductors [1]. Power and skew values are measured using a HSPICE simulation-based parametric analysis. Skew parameters are varied to approximate the possible process variations [165]. Resistance, capacitance and inductance are randomly chosen within ± 10% of the BPTM values for every wire segment of $6k\lambda$. This is done to approximate changes in the width, thickness, height and resistivity of the wires. Transistor effective length, L_{eff}, is varied by 25%. 100 simulations are run at each data point to collect statistically significant values.

Configuration	**Technology**			
	70nm	100nm	130nm	180nm
16 Tile SoC				
Chip Area	4.6mm x 4.6mm	6.6mm x 6.6mm	8.6mm x 8.6mm	11.9mm x 11.9mm
Operating Frequency	5 GHz	2 GHz	1 GHz	500 MHz
Power	31.1 mW	42.4mw	98.6mW	148.7mW
Mean Skew	25.2 ps	29.4ps	56.0ps	40.3ps
Skew Standard Dev.	5.1 ps	7.0ps	13.9ps	9.5ps
Percent Skew	12.6%	5.9%	5.6%	2.0%
64 Tile SoC				
Chip Area	9.2mm x 9.2mm	13.3mm x 13.3mm	17.2mm x 17.2mm	23.8mm x 23.8mm
Operating Frequency	5 GHz	2 GHz	1 GHz	500 MHz
Power	126.2 mW	240 mW	445.1mW	784.5mW
Mean Skew	41.3 ps	49.7 ps	92.2ps	70.6ps
Skew Standard Dev.	6.2 ps	7.4 ps	56.3ps	11.5ps
Percent Skew	20.7%	9.9%	9.2%	3.5%
256 Tile SoC				
Chip Area	18.5mm x 18.5mm	26.4mm x 26.4mm	NA	NA
Operating Frequency	5 GHz	2 GHz		
Power	749 mW	1.29W		
Mean Skew	81.6 ps	89.7ps		
Skew Standard Dev.	22.5 ps	11.6ps		
Percent Skew	40.8%	17.9%		

Table 6.5. Clock Specifications for Various Tile Numbers and Technologies

Table 6.5 clearly shows that global clocking is still practical when used in this tiled architecture. Power, while not low, is well below the 20W used in the Alpha 21164 clock distribution system when operating at 266MHz [70]. Skew is slightly more problematic, as the maximum clock frequency will be limited by the sum of the interconnect critical path and the maximum skew between neighboring tiles. In

the simulations, skew is calculated as the maximum difference between the rising edge of all the possible pairs of clock terminals. No effort is made to identify if the pair with the maximum clock difference are actually neighbors. For aSoC, skew only matters as it applies to neighboring tiles. As such, the likelihood of the worst case skew shown in Table 6.5 occurring between neighboring tiles is increasingly small as the number of tiles increases. Testing each part for skew should result in better-than-predicted performance. Additionally, as this clock system is intended as a simple demonstration of practicality, advanced clocking techniques are not investigated. Using an active de-skew technique could reduce skew by over 70% between neighboring tiles [166].

On a side note, the 256-tile system in the $70nm$ technology has $10Tera$ Bytes/sec of internal network bandwidth. Using the wire capacitance for the 33kλ tile size, the power dissipated in the interconnect wires alone could be as high as 6.7W.

6.2.2.3 Power Distribution

Power supply delivery is made simpler with the uniform floorplan. This is especially important when attempting to deliver multiple supply voltages as shown in Figure 6.7. The hierarchical power distribution system can be designed in advance to supplement the needs of the cores in the SoC library.

Table 6.6 shows the specifications of the example power hierarchy of Figure 6.7. While the power supply distribution structure of Figures 6.4 and 6.7 is simplistic, it does allow for the evaluation of the supply parameters.

The local supply distribution network is aimed at supplementing the internal power distribution system of the cores. As shown in the top portion of Table 6.6, it provides a large amount of additional decoupling capacitance and reduces resistance across the tile. The capacitance added is greater than the gate capacitance of over $500k$ minimum-sized transistors, and so it helps hold the local supply constant

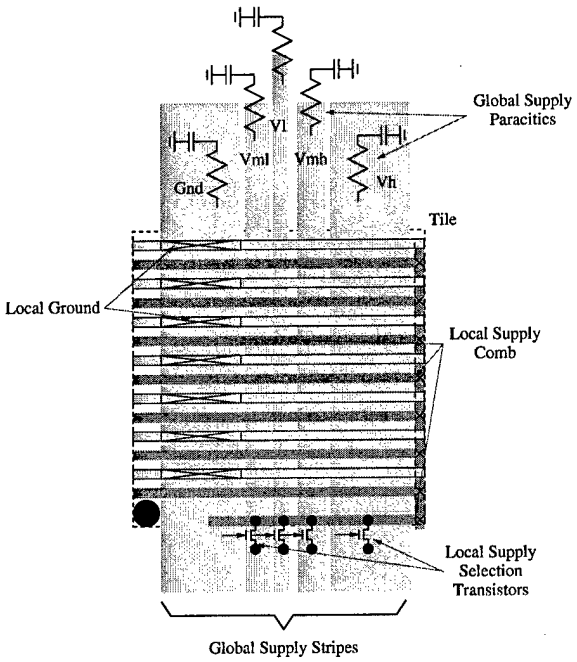


Figure 6.7. Power Grid Allocation

during heavy core activity. Set up in an interlocked comb-like structure, power and ground can be sent across the tile through low resistance stripes. Each of the 16 stripes is capable of delivering 31.6mA of current across the tile with a voltage drop of only 18mV, 1% of a 1.8V supply. This capability is most effectively used when the core internal distribution system is aligned perpendicularly to form a mesh with the local supply comb. The low wire delay associated with the local supply stripes enables quick and uniform voltage scaling across the core.

The global supply is a more complex mesh structure, best seen by viewing both Figures 6.4 and 6.7. It consists of five coarsely weaved power meshes, which carry four values of V_{dd}, (V_h, V_{mh}, V_{ml}, and V_l), and ground to all tiles. The allocation of metal to each mesh is based on the amount of current required. Higher values of V_{dd} with higher core clock frequencies require considerably more current. Table 6.6 attempts to show example specifications for the global supply meshes.

Configuration	Technology			
	70nm	100nm	130nm	180nm
Local Supply 33kλ Tile Size				
Tile Area	1.2mm x 1.2mm	1.7mm x 1.7mm	2.2mm x 2.2mm	3.0mm x 3.0mm
Capacitance	122pF	211pF	273pF	396pF
Transistor Gate Capacitance	.00018pF	.0003pF	.00044pF	.00075pF
Resistance per Supply Stripe (16 Stripe)	.6Ω	.6Ω	.6Ω	.57Ω
Supply Stripe Time Constant	1.73ps	2.97ps	3.84ps	5.33ps
Global Supply 64 Tile SoC (5 Supply System)				
Chip Area	9.2mm x 9.2mm	13.3mm x 13.3mm	17.2mm x 17.2mm	23.8mm x 23.8mm
f_{max}	5GHz	2GHz	1GHz	500MHz
V_h for f_{max}	0.7V	1.0V	1.3V	1.8V
W_h 9330λ	653μm	933μm	1213μm	1679μm
R_h	.11Ω	.11Ω	.11Ω	.11Ω
C_h	4.86nF	8.40nF	10.9nF	15.7nF
$f_{res\ h}$	29.5MHz	22.4MHz	19.7MHz	16.4MHz
$I_{h\ max}$ 2% drop	40mA	58mA	75mA	105mA
$P_{h\ max}$ 2% drop	1.79W	3.7W	6.24W	12.1W
V_{mh} for $2f_{max}/3$	0.45V	0.64V	0.84V	1.16V
W_{mh} 3732λ	261μm	373μm	485μm	672μm
R_{mh}	.28Ω	.28Ω	.28Ω	.28Ω
C_{mh}	1.95nF	3.36nF	4.35nF	6.3nF
$f_{res\ mh}$	46.5MHz	35.4MHz	31.1MHz	25.9MHz
$I_{mh\ max}$ 2% drop	10.5mA	14mA	17mA	25mA
$P_{mh\ max}$ 2% drop	0.3W	0.57W	0.91W	1.86W
V_{ml} for $f_{max}/2$	0.3V	0.43V	0.56V	0.77V
W_{ml} 2799λ	196μm	280μm	363μm	503μm
R_{ml}	.37Ω	.37Ω	.37Ω	.37Ω
C_{ml}	1.46nF	2.52nF	3.26nF	4.72nF
$f_{res\ ml}$	53.8MHz	40.9MHz	36.0MHz	29.9MHz
$I_{ml\ max}$ 2% drop	5mA	7.5mA	9.5mA	13mA
$P_{ml\ max}$ 2% drop	96mW	0.21W	0.34W	0.64W
V_l for $f_{max}/3$	0.24V	0.33V	0.44V	0.6V
W_l 1866λ	131μm	187μm	243μm	336μm
R_l	.55Ω	.55Ω	.55Ω	.55Ω
C_l	0.97nF	1.68nF	2.17nF	3.15nF
$f_{res\ l}$	66.0MHz	50.1MHz	41.1MHz	36.6MHz
$I_{l\ max}$ 2% drop	2.8mA	3.8mA	5mA	7mA
$P_{l\ max}$ 2% drop	43mW	80mW	141mW	269mW

Table 6.6. Example Power Grid Specifications

For each of the four V_{dd} values, the specifications are described in terms of seven factors.

1. Supply Voltage Value V_{dd}, (V_h, V_{mh}, V_{ml}, and V_l): The supply voltage for each mesh is determined using the voltage-delay properties discussed in Chapter 2, Figure 2.25. For this discussion, the core clock frequency can be set to f_{max},

$2f_{max}/3$, $f_{max}/2$, or $f_{max}/3$, which sets $V_h = V_{max}$, $V_{mh} = 0.64V_{max}$, $V_{ml} = 0.43V_{max}$, and $V_l = 0.34V_{max}$

2. Mesh Stripe Width W: The width of each global mesh stripe, given in micrometers, is selected based on projected current requirements and the measured wire resistance.

3. Mesh Segment Resistance R: This value represents the resistance of a tile length segment of the global mesh stripe.

4. Entire Supply Capacitance C: Each supply should have high total capacitance to reduce the effects of charge sharing. This voltage-selective system is especially prone to charge sharing when multiple cores change voltages at the same time. As such, the global mesh capacitance should be supplemented with off-chip capacitors.

5. Power Grid Resonant Frequency f_{res}: The on-chip power grid capacitance when coupled with the package inductance creates an unfortunate resonant circuit in the supply with frequency $f_{res} = 1/(2\pi\sqrt{L \cdot C})$ [167]. The frequencies in this table are calculated using an inductance, L, of $6nH$, as this is representative of the ceramic packages from Kyocera^{TM}used in MOSIS fabrication [164]. Notice that the resonant frequency for the power supply is less, in some cases much less, than the clock frequency. As such, the switching activity of the system may result in power supply resonance. The two-tiered power hierarchy will dampen the overall response, as the resonant frequency for each tile is more than an order of magnitude higher than the global mesh. Serious problems, however, may occur in the case where the dynamic voltage scaling system causes voltage supply switching at rates near the global supply resonant frequency.

6. Average Maximum Core Current I_{max}: Using HSPICE simulations of each mesh, the maximum current delivery properties are determined. For these simulations all 64 cores are assumed to be using the max current, I_{max}. This max current is determined such that the worst case $I_{max}R$ voltage drop in the supply mesh is no more than 2% of the mesh supply voltage, (V_h, V_{mh}, V_{ml}, and V_l). In order to achieve these results, each mesh is connected to eight well-placed external delivery points using bump arrays. As such, the five supply meshes require only 40 bumps to connect to the package. Higher current could be supported if more connection bumps are used.

7. Total Power Capability P_{max}: Under the worst case conditions the global power supply mesh delivers P_{max} to the entire aSoC.

6.2.2.4 Chip Input and Output

In aSoC, input and output (I/O) can be handled as another tile as shown in Figure 6.8. The figure shows tiles, with specialize communication interfaces (CI), connected to arrays of solder bump pads. I/O tiles are shown on the edge of the chip, but complete rows or columns of I/O tiles could be used in the interior of the system.

If the I/O is constructed as a tile, the core is replaced with the I/O pads and the asynchronous protocol discussed in Chapter 4 can be used for off-chip communications. As such, much of the CI infrastructure can be reused. The construction of the I/O tiles would require some special features. First, the need for pads means that no global power stripes can pass over the tile. Second, the overhead of dynamic voltage scaling is not required as the I/O should always use the highest supply on-chip. Third, as always, special buffers will have to be used to drive or receive off-chip signals.

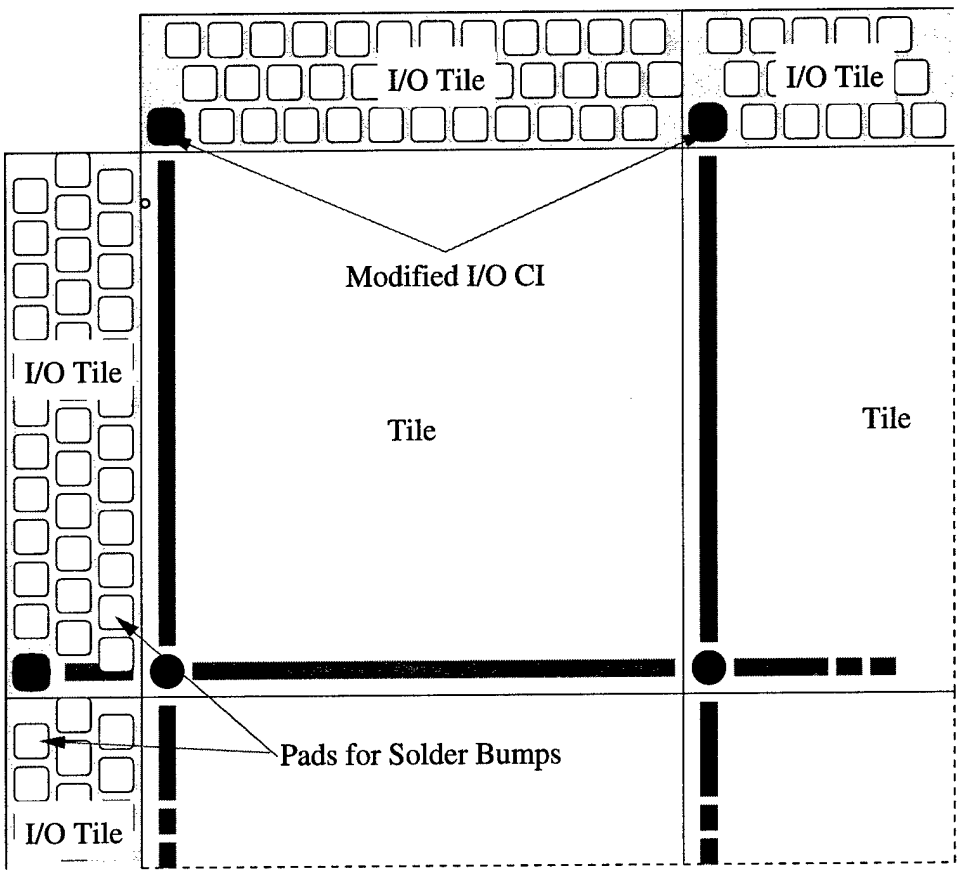


Figure 6.8. Input and Output Tiles

6.2.3 Communications Interface Development and Evaluation

The design flow for the infrastrucure development, shown in Figure 6.9, is developed to allow complete logical and functional verification of the aSoC layout prior to fabrication. The complete design is broken up into a hierachy of subsystems, and each subsystem is modeled logically in verilog, and functionaly as a schematic and layout. The verilog models and test bench establish and test the desired behavior of each subsystem. A schematic is then generated and extracted. The extracted schematic can be simulated in HSPICE to evaluate functionallity and size transistors. Using *Perl* scripts, the extracted netlist can be converted to a verilog netlist using the *nmos*, *pmos* and *trans* functions. This verilog netlist can be simulated using the original verilog test bench to verify logical behavior. From the

schematic, a custom layout is designed. Layout vs. schematic comparison tools are used to logically verify the circuit. The layout is then extracted with paracitic resistance and capacitance for simulation in HSPICE. The HSPICE simulation confirms the final functionality of the system. The layout is completed using MOSIS deep sub-micron scalable design rules [164] compatible with both the TSMC $250nm$ and $180nm$ processes.

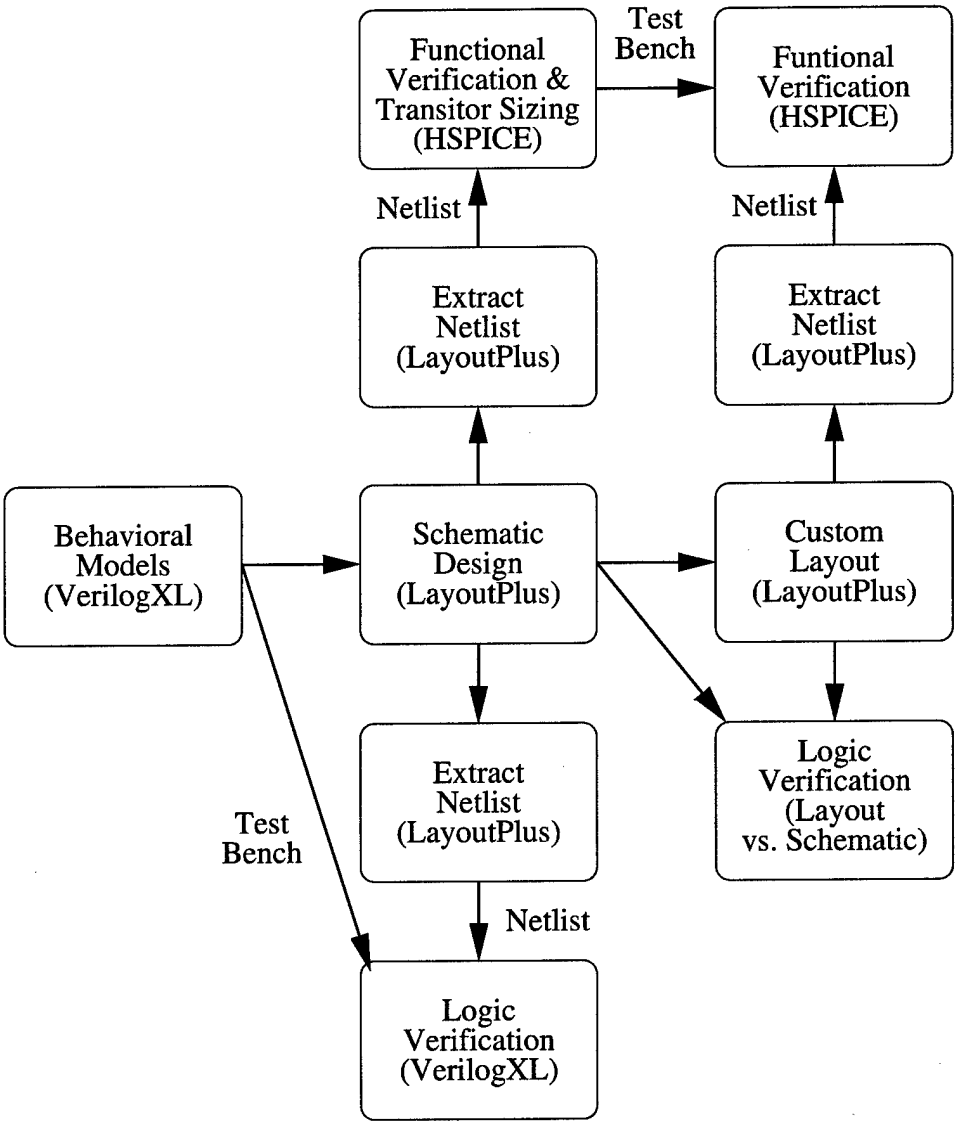


Figure 6.9. Custom Layout Design Flow

The architecture of the CI is discussed in detail in Chapter 4. For implementation it is convenient to break the CI into four subsystems as shown in Figure 6.10.

1. Communication Controller and Instruction Memory +
2. Crossbar and Data Transfer
3. Core-Ports
4. Frequency and Voltage Scaling System

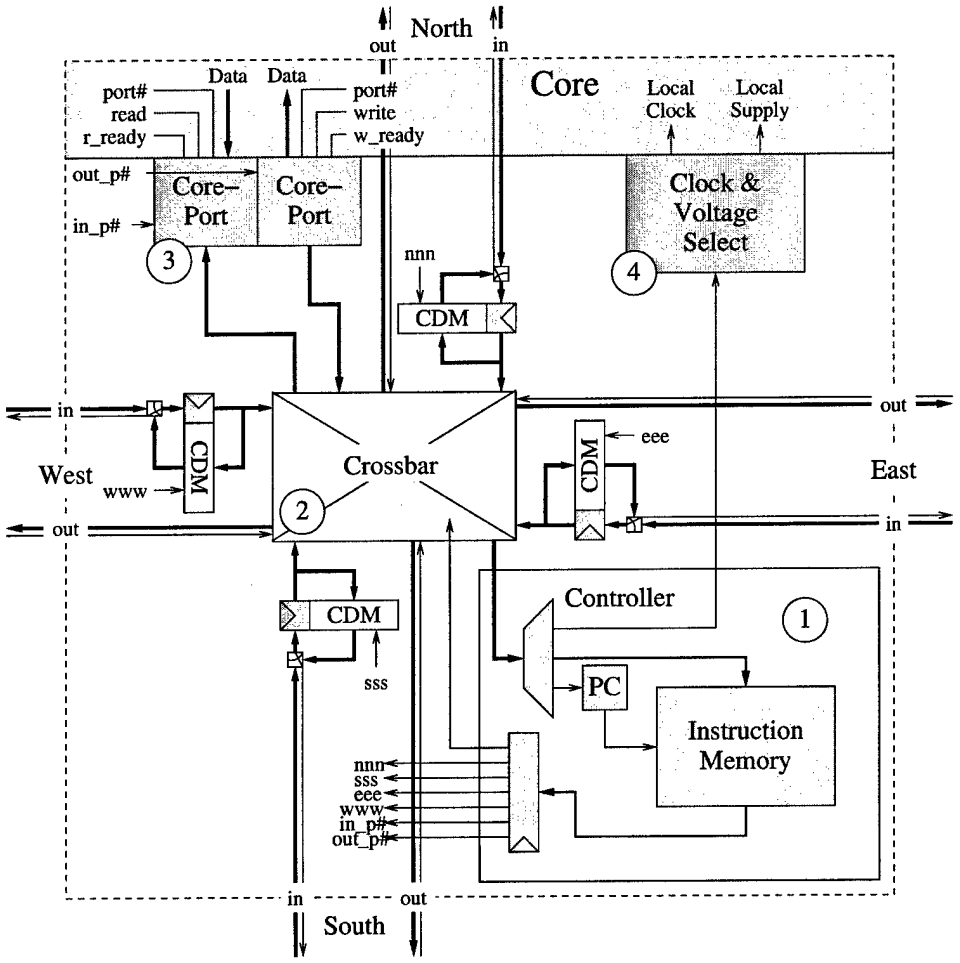


Figure 6.10. Communication Interface Block Diagram

The implementation of each of these subsystems is discussed in the following sections. Then the final implementation is presented in the form of a simple test chip.

6.2.3.1 Communication Controller and Instruction Memory (CCIM)

The layout of the CCIM is shown in Figure 6.11 The controller portion resides in the center of the layout, aligned horizontally. The memory structure is subdivided into two banks of 16 words each with the decoders for each bank located in the center, vertically.

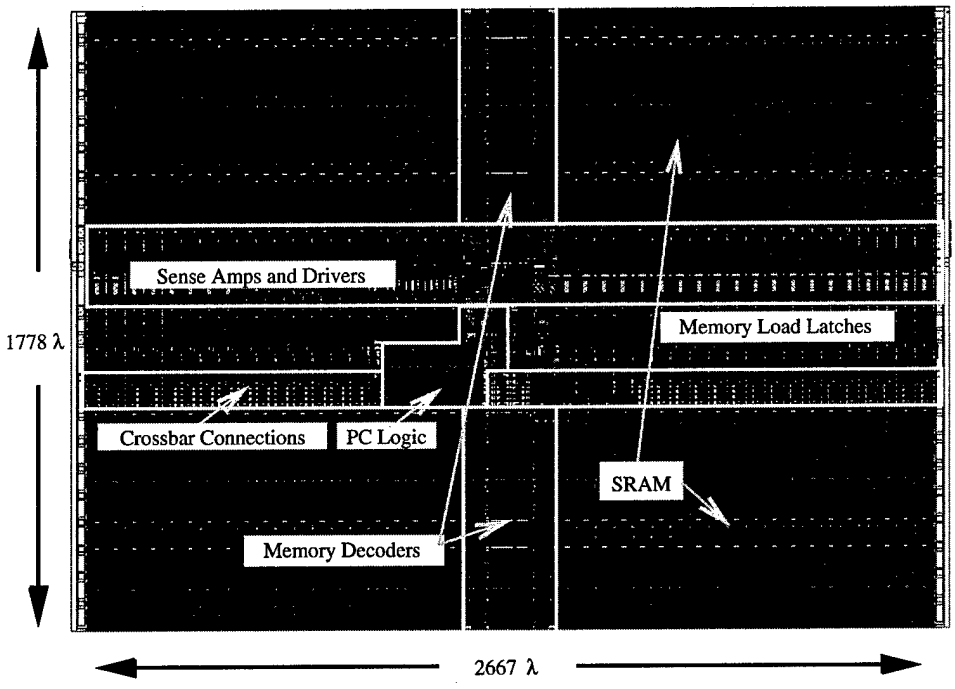


Figure 6.11. Controller and Instruction Memory Layout

An HSPICE model developed based on extracted parasitic layout was extracted and tested in HSPICE using BPTM [68]. Table 6.7 lists some of the important specifications for the CCIM in 180nm technology The system is relatively small and is dominated by the SRAM-based instruction memory. As such, the CCIM

Number of transistors	12594
Size in λ	$2667\lambda \times 1778.5\lambda$
Size in Microns	$240\ \mu \times 160\mu$
Critical Path Delay	600ps
Maximum Clock Frequency	833MHz
Power (500MHz)	5.7mW

Table 6.7. CCIM Layout Specifications

size can be scaled by changing the number of available communication instructions. The critical path for the entire controller portion is the read, which involves the decoder select, the word line, the bitline and the sense amp, as shown in Figure 6.12. The instruction memory is a dual-ported SRAM containing 32, 40-bit words and is organized as shown in Figure 6.12. The memory uses a typical precharge and read approach and therefore must traverse the critical path in half a cycle. Therefore the maximum clock frequency in Table 6.7 is half the frequency of the critical path.

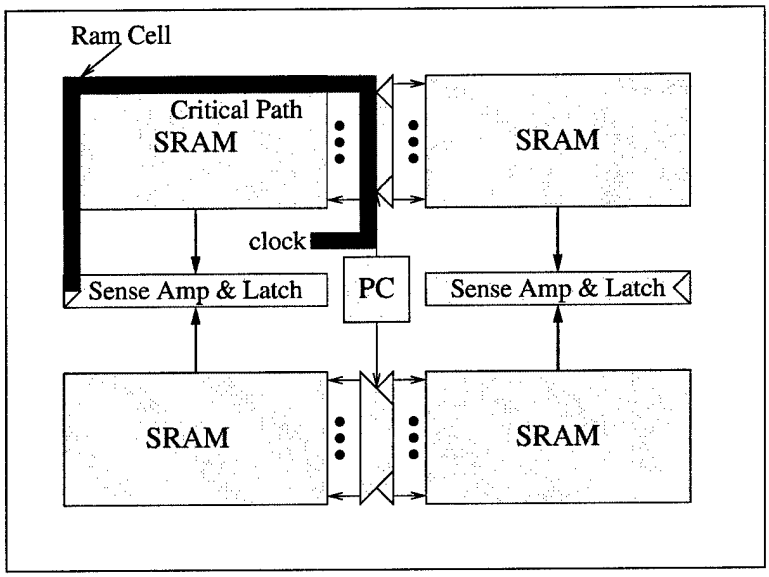


Figure 6.12. Instruction Memory Block Diagram

Power consumption is dependent not only on the clock frequency but the data contained in the instruction memory. The number presented in Table 6.7 represents a average case of data and clock based on the architectural work in [37]. In this work,

aSoC is run at 500MHz, and each communication interface supports an average of two streams per cycle.

6.2.3.2 Crossbar and Data Transfer System

Figure 6.13 shows a block diagram of the communications between two CIs. As discussed in Chapter 4, inter-tile communications happens in two cycles. First, the communication data memory is accessed to see if data has been held from the last transfer. If data has been held, it will be sent in place of any incoming data. Second, the crossbar is set and data goes through the crossbar across the interconnect to the next tile. At the same time data is being sent, the flow control bit for that transfer is being sent backwards from the receiving tile. This bit indicates the pending success of the transfer. If the transfer is not successful, the data must be buffered in the sender's CDM. If the transfer is successful, the flow-control bit clears the sender's CDM buffer. The flow-control bit represents the critical path of the transfer as it must come directly from the receiver's CDM in the transfer cycle. This is highlighted in Figure 6.13.

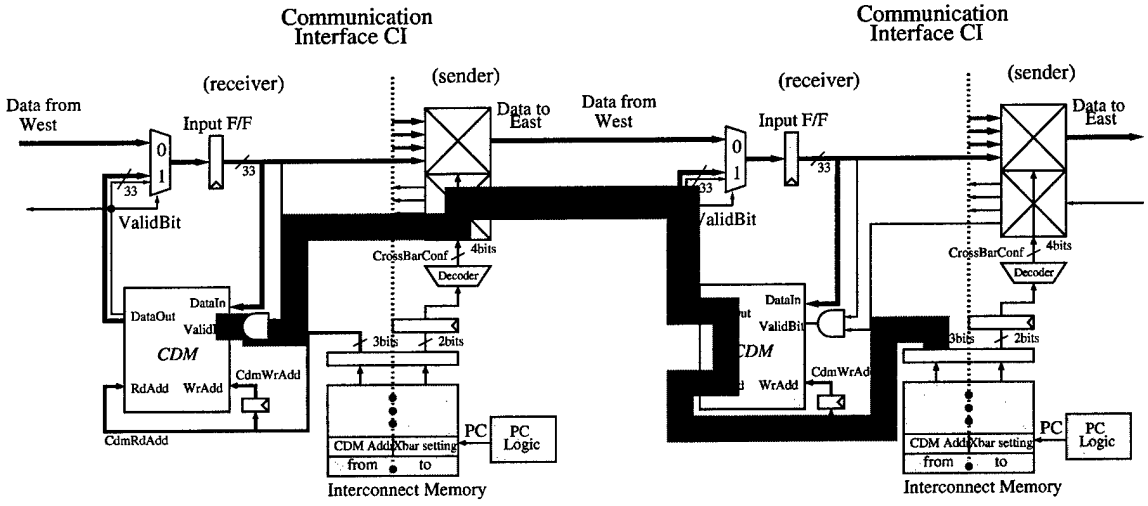


Figure 6.13. Communication Block Diagram

Figure 6.14 shows a layout of the CDM, including the *interconnect drivers*, and *crossbar connections*. Data travels vertically in the figure with the intra tile interconnect on top and the crossbar on bottom. The full communication interface requires four of these systems, one for each direction.

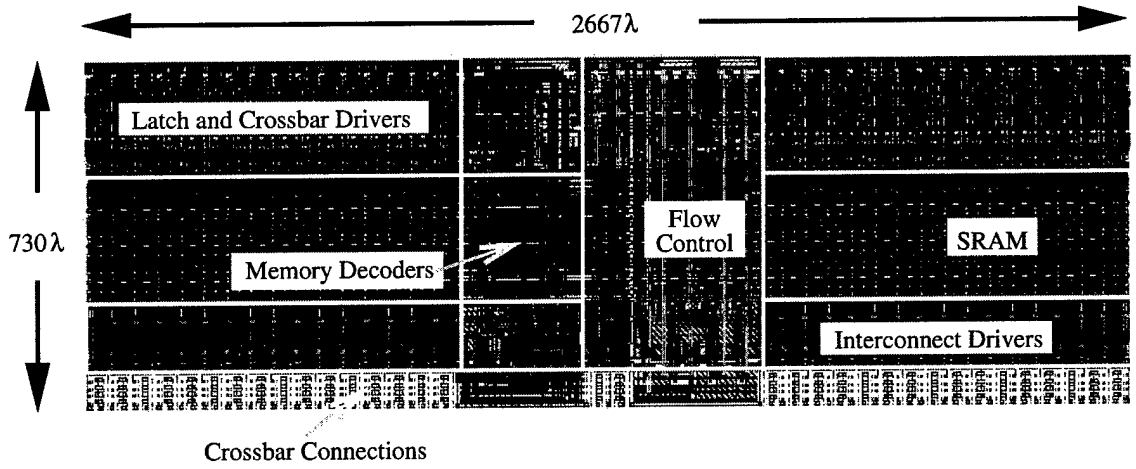


Figure 6.14. One Side of the Communication Data Flow Layout

As with the CCIM the layout is extracted and used in Hspice simulations to evaluate the design. The results are show in Table 6.8. Both the critical path of the flow-control bit, shown in 6.13 and a normal data bit are given. From this data, it can be seen that the addition of flow-control memory access to the critical path adds nearly 300ps of delay. This means that there exists almost 300ps of slack in the transmission of data from one tile to the next. Actually, there is close to 400ps of slack when the critical path of the CCIM is considered. This creates an opportunity for power savings in the communication interface. The total power of the CCIM is dominated by the cost of transmitting data. In fact, per side data transmission accounts for nearly 80% of the power used. Simply driving the interconnect with the next lower available voltage, 1V in this system, reduces the data bit power to .607mW and only increases the delay to 1.12ns. This cuts the per side data transmission power to less than 15mW, with no overhead in area or delay.

Number of transistors	4286 (One Side)
Size in λ	$2667\lambda \times 730.5\lambda$
Size in Microns	$240\ \mu \times 66\mu$
Critical Path Delay (30k λ wire)	
Flow-Control Bit	1.07ns
Data Bit	0.81ns
Maximum Clock Frequency	935MHz
Power per Bit (500MHz)	
Flow-Control Bit	1.6mW
Data Bit	1.02mW
Overhead Power	¡ 3mW
Total Power per Side (50 % Data Activity)	22mW

Table 6.8. Data-Flow Layout Specifications

	Input Port	Output Port
Number of transistors	1825	3155
Size in λ	$2667\lambda \times 278\lambda$	$2667\lambda \times 333\lambda$
Size in Microns	$240\ \mu \times 25\mu$	$240\ \mu \times 25\mu$
Power (500MHz and 50% switching)		
Read	2.4mW	4.4mW
Write	3mW	2.4mW
Total Power	5.4mW	6.8mW
Critical Path Delay	NA	1ns

Table 6.9. Core-Port Specifications

6.2.3.3 Core-Ports

The core-ports are responsible for buffering data as it crosses the frequency and voltage boundary between the core and CI. The asynchronous hand-shaking protocol used to connect the core to the interconnect is discussed in detail in Chapter 4. The functionality of this protocol has been evaluated in both verilogXL and HSPICE.

From the interconnect side of the interface the core-ports are identical to the CDM. As such, the core-port critical path is identical to the one shown in Figure 6.13 with the CDM replaced by the core-port buffer. From a layout perspective, the buffer memory and decoders used in the CDM can be re-used for the core-ports. With this in mind the critical specifications of the core-ports are given in Table 6.9.

The power in Table 6.9 assumes that data is being both sent and received from the core. When evaluating a system architecture the total power can be divide by the number of cycles between transfers.

To traverse the voltage difference, a simple yet effective circuit is used, as shown in Figure 6.15 [168]. The circuit consist of three parts: the *Low Voltage Driver*, which creates a differential voltage; the *High Voltage Receiver*, which contains two strong cross-coupled pull down circuits; and the *SR Latch*, which sharpens the signals.

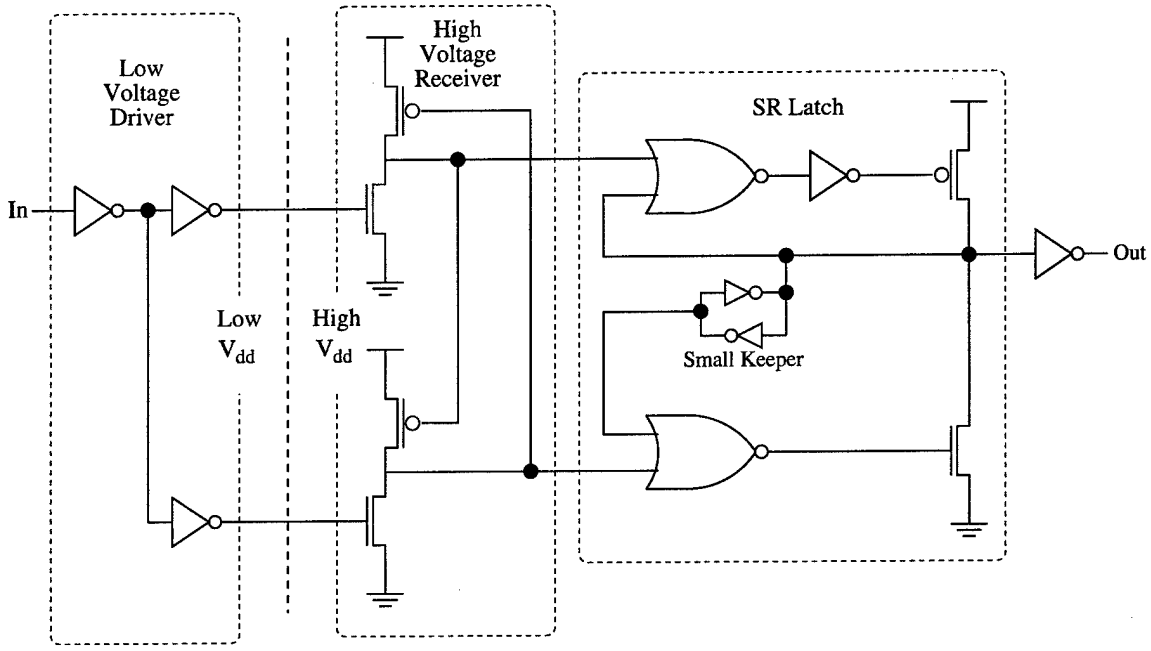


Figure 6.15. Voltage Interface

The delay of across the voltage interface circuit is the critical path of the core-ports when going from the minimum voltage, .65V, to the maximum voltage 1.8V. This only happens in the output core-port as the CI is assumed to be at the maximum voltage always. The delay of 1ns is comparable to the other delays of the system.

6.2.3.4 Clock and Voltage Selection System

An example voltage selection system has already been discussed in Chapter 2. Figure 6.16 shows the resulting layout of output transistor in a 180nm process. Even though the final PMOS width is $5.3mm$, the transistors can be folded so

Number of tiles	9	16
Power	239mW	424mW
System Bandwidth	64GB/sec	36GB/sec

Table 6.10. Example System Specifications Using Implementation Results in a Calculator

the entire multi-stage driver and output pull-up can fit within $70\mu m \times 105\mu m$. This corresponds to $776\lambda \times 1168\lambda$. The circuit was extracted and simulated with HSPICE. The delay in charging the entire local supply comb is only 1ns.

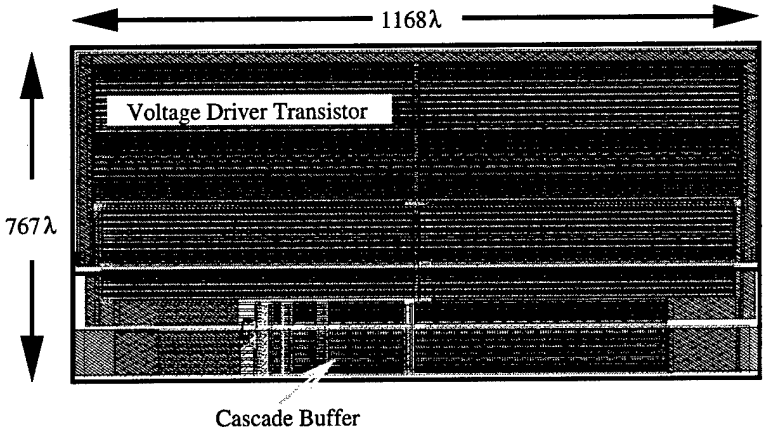


Figure 6.16. Voltage Interface

6.2.3.5 Implementation Summary

The data collected from the above implementations can be used in architectural-level simulations to provide accurate timing and power data. Table 6.10 shows the power consumption for several aSoC architectures assuming average transmission frequency [37].

Chapter 7

Architectural Results of Voltage Scaling

This chapter identifies the characteristics of applications which make the system-wide dynamic voltage-scaling approach feasible. The system described in Chapter 5 requires streaming applications and dynamically parameterized cores. Streaming applications establish expected communication patterns. It is the deviations from these patterns which indicate problem cores. The motivation for dynamic voltage scaling is based on the notion that the processing throughput of the cores varies at run-time. Dynamically parameterized architectures and applications create these run-time variations.

Although extensive evaluation of benchmark applications is beyond the scope of this document, several small test cases are used to demonstrate the functionality of the system-wide voltage and frequency scaling approach.

7.1 Target Application Characteristics

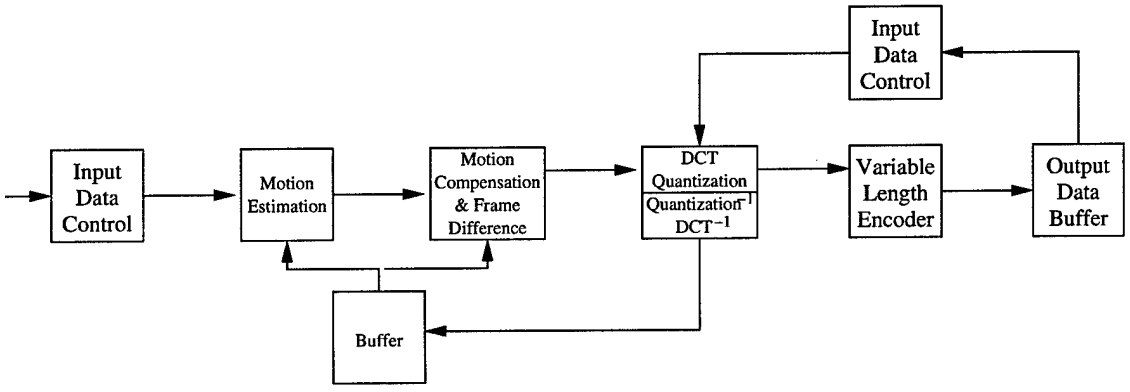


Figure 7.1. MPEG Encoder Block Diagram

For six reasons, MPEG and other DSP applications are ideal SoC system-wide voltage and frequency scaling.

1. The video encoder uses several distinct steps in processing the input data. Figure 7.1 shows a partitioning of the MPEG encoder onto the aSoC tiled architecture. As such, these processing steps provide initial partition information for mapping the encoder to aSoC. Additionally, the implementation of each processing step can be performed and optimized with some level of independence. This allows for the independent development and parameterization of individual cores.

2. The encoding process works on streaming input video data. Data comes in at specified rates and leaves the system with specific time constraints. Within the system, data proceeds through the processing steps with strict regularity. This data regularity invites the use of static scheduling of interconnect bandwidth, while the time constraints support frequency scaling.

3. By design, the MPEG standards are extremely flexible. The internal processing can be performed in any way, providing the output bit stream is compliant. This gives the SoC designer a great deal of flexibility to evaluate system design trade-offs, and incorporate *dynamic parameterization.*

4. Output precision or quality can vary. Often in communication systems quality degradation is imposed by the system environment. The ability to use quality as a parameter or metric enhances the flexibility of the system and allows for further *dynamic parameterization.*

5. Although the system communications are well behaved, core loading may vary dramatically over the life of the application. This could be the result of input data properties or parameter adjustment. The use of *dynamically*

parameterized cores increases the likeliness of variations in core utilization. It is these variations in core utilization that are exploited by the voltage selection method.

6. Finally, and possibly most importantly, these properties characterize nearly all communications applications and many data acquisition and feedback control problems. Performing a detailed analysis on MPEG encoding will give valuable insight on the feasibility of this approach for an expansive and important subset of computing problems. It is important to note that MPEG is not necessarily representative of computer-aided design, optimization, data-base management and other scientific computing problems. These are therefore excluded from consideration.

7.2 Methodology

To evaluate frequency and voltage scaling, several simple systems can be modeled. Although complete SoC implementation is beyond the scope of this work, specific test cases can be modeled in C and simulated using the aSoC network simulator. There are two phases to this project.

1. Modification of simulator to handle voltage scaling

2. Simulation and results collection

7.2.1 aSoC Simulator Modifications

The system evaluation is centered on a cycle-based simulator for aSoC developed by J. Liang [36]. This is a version of the NuMESH simulator, nsim [169], modified to fit the flow control protocol of aSoC. For the purpose of this discussion, the simulator can be described as having two main parts: the network model and the core models. The network model provides an environment for the simulation of code in the core

models. This model tracks timing and executes core functionality when necessary. The network model provides connectivity between the cores using a static schedule developed automatically by the aSoC compiler, AppMapper [39]. The core models implement the algorithm. They are responsible for internal timing through the use of cycle delay statements.

To simulate the frequency and voltage scaling system, the simulator must be modified to contain the frequency and voltage controller discussed in Chapter 5. Additionally, each core must be modified to respond to the frequency and voltage controls. All of this functionality can be added to the core models. The controller can monitor the core-ports from the core side of the interface. Detection of blocked streams is analogous to the system discussed in Chapter 5. Frequency modification is simple, as it only requires changing the cycle delay statements. Power estimation is based on the voltage delay characteristics discussed in Chapter 2.

7.3 Example Systems

To evaluate the frequency and voltage selection architecture of Chapter 5, several small core scenarios are simulated. In each case the thresholds for frequency and voltage scaling are as shown in Table 7.1. The global interconnect is the highest frequency in the system and the cores can not have a throughput faster than one transfer per global cycle. When voltage is increased it causes a 10-cycle penalty. Decreasing the voltage only incurs one cycle. All cores are assumed to use the same power when running at their fastest possible throughput. Four levels of voltage are available. The three lower levels correspond to 1/2, 1/4 and 1/8 the max speed of the core and result in 70%, 86%, and 89% reduction of core power. The 2% power penalty of the voltage scaling system is neglected. Additionally, the interconnect power is neglected for simplicity. With all of this the power numbers in this section indicate the weighted percent of computations using low voltage and frequency.

Parameter	Value
Th_{down}	20 Output Fails
Th_{up}	20 Input Fails
$Th_{\overline{down}}$	10 Input Fails
$Th_{\overline{up}}$	10 Output Fails

Table 7.1. Values for Core Utilization Measurement Thresholds

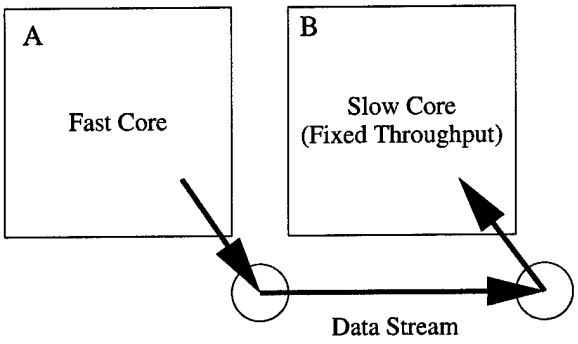


Figure 7.2. Fast Core to Slow (Bottleneck) Core

7.3.1 Test 1: Fast Core, Fixed Slow Core

Figure 7.2 shows a possible combination of cores, where a source core, A, has 16 times the throughput of the destination core, B. In this case, core B's fastest possible throughput is 1/16 the global clock. The source core can run at 1/2 transfer per global cycle, and it starts with this speed.

A stream of data consisting of 1000 transfers is sent from core A to core B. System performance is evaluated both with and without automated scaling. Table 7.2 shows the resulting specifications for the system and Table 7.3 shows the scaling activity for each core.

With no loss in performance, the frequency of core A automatically drops to match the throughput of the bottleneck. The system settles to the best case frequency before the 10th data transition in the stream, and as a result achieves nearly the best-case power.

Parameter	No Scaling	Scaling
# of cycles	16027	16027
System Power	2	1.113

Table 7.2. Core Slow Down and Resulting Power Savings due to Stream Bottleneck

	Core A	Core B
Start Frequency	1/2	1/16
End Frequency	1/16	1/16
Transitions	1/2 to 1/4 at cycle 37	-
	1/4 to 1/8 at cycle 67	-
	1/8 to 1/16 at cycle 119	-

Table 7.3. Frequency and Voltage Transitions in Dynamic System

7.3.2 Test 2: Fixed Fast Core, Slow Core

Figure 7.3 shows a complementary example to the one described above. In this case, some design constraint forces core A to run with a throughput of 1/2 transfer/cycle. Core B starts with a low throughput of 1/16, but is now allowed to scale.

Again a stream of data consisting of 1000 transfers is sent from core A to core B, and the system performance is evaluated both with and without automated scaling. Table 7.4 shows the resulting specifications for the system and Table 7.5 shows

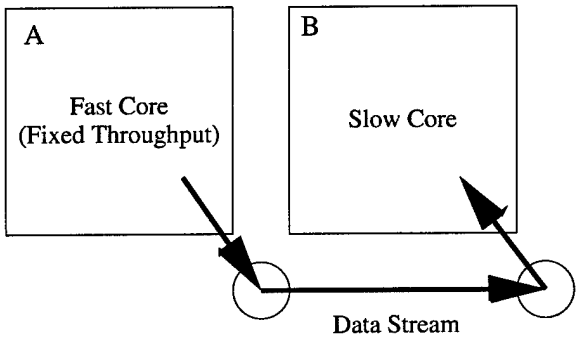


Figure 7.3. Fast Core Drives Performance Up

Parameter	No Scaling	Scaling	Best Case Performance
# of cycles	16027	2131	2041
System Power	1.11	1.97	2

Table 7.4. Core Speed Up and Resulting Performance Increase

	Core *A*	Core *B*
Start Frequency	1/2	1/16
End Frequency	1/2	1/2
Transitions	-	1/16 to 1/8 at cycle 20
	-	1/8 to 1/4 at cycle 62
	-	1/4 to 1/2 at cycle 102

Table 7.5. Frequency and Voltage Transitions in Dynamic System

the scaling activity for each core. In Table 7.4 a column is added to show the performance when both cores start at a throughput of 1/2.

In this system the frequency of core *B* increases quickly to 1/2, to achieve good performance. As such, the power is nearly the same as the best performance case. The power presented here is somewhat misleading, as only successful transfers are attributed with using power. The *no scale* case takes nearly eight times the number of transfer attempts and should therefore have worse-than-predicted power.

7.3.3 Test 3: Fast Core, Variable Core

Figure 7.4 shows a combination of cores, where a destination core experiences a period of slowdown and then recovers. Both cores start with throughputs of 1/2, but core *B* slows down to 1/16 after 250 transfers, then recovers again after 500 transfers.

A stream of data consisting of 1000 transfers is sent from core *A* to core *B*. System performance is evaluated both with and without automated scaling. Table

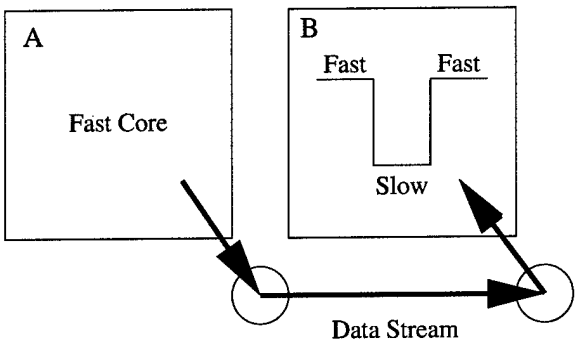


Figure 7.4. Fast Core Connected to Core with Slowdown

Parameter	No Scaling	Scaling
# of cycles	5541	12426
System Power	2	1.15

Table 7.6. Scaling Creates New Bottleneck

7.6 shows the resulting specifications for the system and Table 7.7 shows the scaling activity for each core.

When core B has a slowdown, core A adapts to a lower voltage frequency setting in an attempt to save power. With no upstream core to drive core A back up it is stuck in the low power mode and becomes bottleneck to computations.

7.3.4 Test 4: Variable Core, Slow Core

Figure 7.5 shows yet another example system. In this case, core A executes some code faster than usual. After 250 transfers its performance jumps from 1/16

	Core A	Core B
Start Frequency	1/2	1/2
End Frequency	1/16	1/2
Transitions	1/2 to 1/4 at cycle 548	1/2 to 1/16 at cycle 515
	1/4 to 1/8 at cycle 580	1/16 to 1/2 at cycle 4525
	1/8 to 1/16 at cycle 628	-

Table 7.7. Frequency and Voltage Transitions in Dynamic System

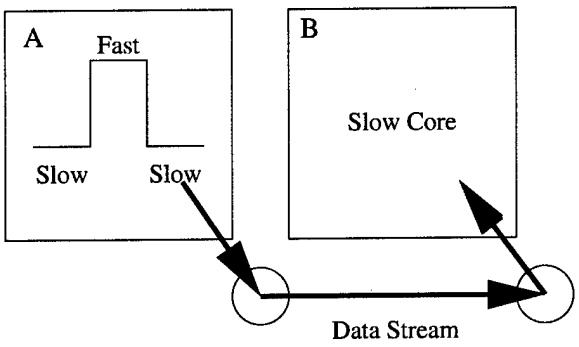


Figure 7.5. Performance Spike Drives Core to Higher-than-Needed Voltage

Parameter	No Scaling	Scaling
# of cycles	16027	12611
System Power	1.11	1.71

Table 7.8. Core Speed Up and Resulting Performance Increase

to 1/2. It returns to the slower mode after transfer 500. Core B starts with a low throughput of 1/16, and is allowed to scale.

Again a stream of data consisting of 1000 transfers is sent from core A to core B, and the system performance is evaluated both with and without automated scaling. Table 7.8 shows the resulting specifications for the system and Table 7.9 shows the scaling activity for each core.

In this case the down stream core B is stuck in a higher speed mode even though the upstream core can not support it. As a result, energy is wasted.

	Core A	Core B
Start Frequency	1/16	1/16
End Frequency	1/16	1/2
Transitions	1/16 to 1/2 at cycle 4026	1/16 to 1/8 at cycle 4053
	1/2 to 1/16 at cycle 4620	1/8 to 1/4 at cycle 4091
	-	1/4 to 1/2 at cycle 4141

Table 7.9. Frequency and Voltage Transitions in Dynamic System

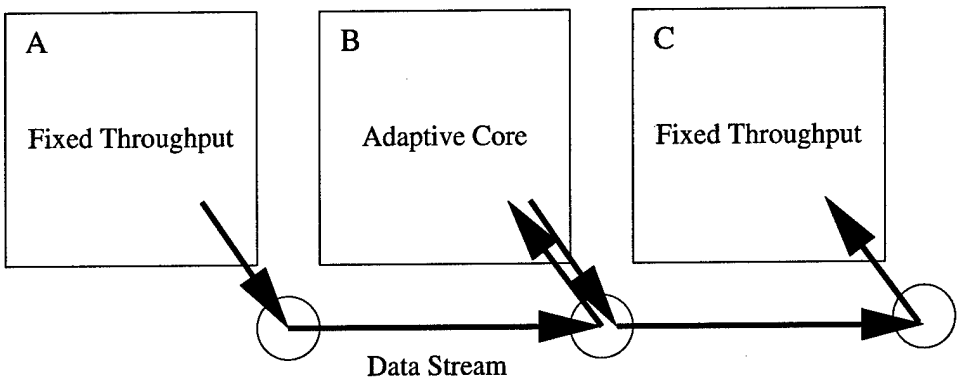


Figure 7.6. Three-Core Test System

7.3.5 Three Core Systems

The preceding simple tests demonstrate some of the key functionalities of the frequency and voltage scaling approach. Using this approach, a core can be slowed down by downstream bottlenecks and sped up by upstream rushes. Once slowed, it can not be sped up until forced by an upstream rush. Likewise, a core which is made to run more quickly can only be slowed by a downstream bottleneck. A simple three-core system, shown in Figure 7.6, can be examined to further demonstrate this interaction. Several tests are outlined in Table 7.10. In all cases cores A and C have fixed rates, while core B is allowed to vary.

In Table 7.10, tests 5 and 7 are simply used to find baseline values for the system, running at speeds of 1/8 and 1/32 respectively. Test 6 is interesting in that an opportunity to slow down is missed. The input core-port of core B never fails as the core is running faster than the upstream core, A. The output core-port of core B never fails as the data rate through the core is limited by the speed of data coming from core A. Test 8 shows a core which should speed up to match the speeds of its neighbor cores. In this case the core overshoots and ends up running twice as fast as necessary. To avoid overshoot, the thresholds for transition could be increased. As with other control systems, dampening overshoot makes the initial response slower as well. Finally, test 9 shows a difficult case for the approach. The downstream

Test #	5	6	7	8	9
Rate (Core A)	1/8	1/8	1/32	1/8	1/2
Rate (Core C)	1/8	1/8	1/32	1/8	1/32
Start Rate (Core B)	1/8	1/2	1/32	1/32	1/32
End Rate (Core B)	1/8	1/2	1/32	1/4	1/8
# cycles	8005	7999	32005	8039	32039
power	2.16	3	2.11	2.297	2.14
Transitions					
				1/32 to 1/16 at cycle 32	1/32 to 1/16 at cycle 26
				1/16 to 1/8 at cycle 68	1/16 to 1/8 at cycle 62
				1/8 to 1/4 at cycle 110	1/8 to 1/4 at cycle 104
					1/4 to 1/8 at cycle 155
					1/8 to 1/4 at cycle 209
					1/4 to 1/8 at cycle 268
					1/8 to 1/16 at cycle 1457
					1/16 to 1/8 at cycle 1481
					1/8 to 1/16 at cycle 1588
					1/16 to 1/8 at cycle 1612
					...

Table 7.10. Frequency and Voltage Transitions in Dynamic System

core is running very slowly while the upstream core is producing data at top speed. The adaptive core, B, adjusts its frequency to approximately the midpoint of the extremes. In doing this the core voltage and frequency oscillates between 1/8 and 1/16. Again higher thresholds will help reduce this oscillation at the cost of less sensitivity.

As a final test, the three-core system is implemented as shown in Figure 7.7. In this case all three cores are initially running at the top throughput. After receiving 250 transfers, the last core experiences a slowdown which lasts for the next 250 transfers.

The result is that core C slows down at cycle 509. Core B has its frequency and voltage dropped twice at cycles 545 and 609. Core C speeds back up at cycle 8509, which allows core B to return to its original speed by cycle 8574.

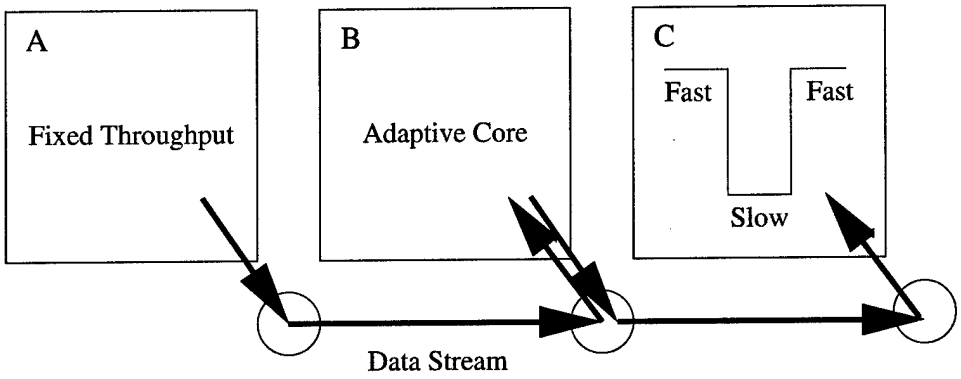


Figure 7.7. Three-Core Test System with Dynamic Throughput Variation

Chapter 8

Conclusions and Future Work

This document strives to identify and resolve several key issues in the development of SoCs. Specifically these issues are:

- **Design time** - aSoC reduces design time through the use of pre-designed intellectual property components. These components include both the processing cores and the interconnect fabric. Chapter 4 provides a background and motivation for tiled architectures that consist of reusable processing cores and interconnect. This dissertation focuses on proving the feasibility of this approach through the modeling and characterization of their reusable interconnect fabric. Chapter 6 demonstrates this feasibility through the presentation of layout level models of key interconnect components.

- **Performance** - J. Liang has investigated various SoC architectures to evaluate performance [34]. As such, this dissertation only evaluates the performance of the interconnect fabric. The layout-level models provide a good tool for evaluating the speed and power consumption of the aSoC interconnect. HSPICE simulation results, shown in Chapter 6, prove that a fast and power-efficient point-to-point interconnect fabric can be developed for the statically scheduled aSoC architecture.

- **Deep Sub-Micron Challenges** - Deep sub-micron design issues, especially clock distribution, power distribution, and global wiring, could greatly reduce the effectiveness of future SoC architectures. This dissertation shows that the use of a rigid SoC floor-plan greatly reduces the impact of deep sub-micron

design problems. The rigid floor-plan is established in Chapter 6 through the creation of a uniform interconnect mesh. Heterogeneous cores of varying sizes can be used in this mesh provided their aspect ratios can be controlled as shown in Chapter 6. Simple models and test circuits of the clock distribution, power distribution, and global wiring systems are constructed and evaluated to show the benefits and cost of using a rigid floor-plan.

- **Power** - This dissertation provides a hierarchical approach to power conservation in SoC. At the base of this approach is the development of dynamically parameterized cores. Chapter 3 presents a background of power conservation approaches and proposes a detailed methodology for the development of cores, which can automatically modify processing in response to various stimuli. This approach is tested in the development of a dynamically parameterized motion estimation architecture. At the system level, I propose a novel interconnect-centric approach for run-time adaptation of core voltage and frequency. With dynamic parameterization, the throughput of each core in the system varies throughout the life of the application. This creates slack and congestion even within a statically scheduled interconnect structure. By monitoring the interconnect, the SoC architecture can track sources of slack and congestion, and automatically modify the voltages and frequencies of the appropriate cores. This is demonstrated for a small representative example application in Chapter 7. This approach supports the modularity and hardware methodology required to address both design time and deep sub-micron issues. Additionally, the choice of discrete voltage selection over continuous voltage scaling limits the performance impact dramatically, as shown in Chapter 2. As such, this approach is ideal for scalable SoC systems.

The approach of this document and the contribution of this dissertation crosses the boundaries of architecture methodology and physical design. As such, the main contributions are three-fold.

1. A scalable hardware speculative approach to power-aware SoC: This work shows a hierarchical approach to run-time power management in SoC, where subsystems are first made power-aware and then complemented by a global voltage scaling scheme. This approach leverages the existing hardware demands of heterogeneous SoC to implement a low-overhead voltage and frequency scaling circuit at each core interface. This approach speculates core utilization at run-time by measuring blockages in the interconnect. Additionally, the system supports the use of algorithm and compiler information through an interconnect interface to each interface controller. This makes it possible to change the behavior of the hardware-only approach.

2. A formalization and example application of *dynamic parameterization* as a method for autonomous run-time power reduction in SoC cores: A detailed methodology for *dynamic parameterization* is presented in the context of other run-time power conservation techniques. This methodology is tested in the development of a dynamically parameterized MPEG motion estimation system. P. Jain (thesis 2001) [23] created a soft core ME device with the flexibility to explore the parameter space of motion estimation. As a result, his work produced the parameter trade-off map for ME. This work adds run-time speculation to take advantage of the trade-offs in ME.

3. A hardware proof of concept for aSoC and a detailed hardware-centric SoC design methodology: My work clearly demonstrates the feasibility of aSoC implementation, and specifically addresses system interconnect, clocking and power supply design. Additionally, key components of the network are designed

in layout to evaluate size, performance and power consumption. As a result, This work validates the concept of aSoC at the hardware level and provides meaningful power and timing data to the architects. In the development of this model a SoC design methodology is developed, which specifically addresses hardware design issues.

In a sense, the present status of the work shows the feasibility of the SoC design concepts presented in this dissertation. As such, the proposed work refines and strengthens the justifications of these concepts. Specifically, two important areas need further refinement in future work.

1. To a large extent the architectural study of Chapter 7 pulls together test cases for the clock and voltage scaling system. At present the aSoC simulator [37] has been modified to test the functionality of the clock and voltage scaling system. To truly test the proposed system applications, which can use voltage scaling, must be mapped to aSoC.

2. Custom layout models of many of the key components in the aSoC interconnect fabric exist. To provide credibility of the physical implementation these should be integrated into a single module.

This dissertation crosses the boundaries of three expansive research areas: SoC architecture, system power management, and deep sub-micron design. As such, it provides a unique broad-based perspective to the development of future SoCs. It presents SoC design concepts including scalable voltage scaling, core dynamic parameterization, and a tile-based design methodology. Then it justifies these concepts with carefully constructed examples, MPEG, motion estimation, and the aSoC hardware design respectively. Although the examples are representative of the systems and applications in the SoC design space, they do not provide a detailed

understanding of the entire design space. This broad-based view opens up many possibilities for future work, including the following:

- Apply voltage scaling to various other SoC architectures.
- Develop more dynamically parameterized systems.
- Develop more aSoC applications to evaluate voltage scaling.
- Apply various signaling techniques to aSoC interconnect and clock generation.

References

[1] *International Technology Roadmap for Semiconductors.*

[2] Sinha, Manoj and Burleson, Wayne. Current-sensing for crossbars. In *Proceedings, IEEE International Application Specific Integrated Circuits/System-on-a-Chip Conference*, Washington, DC, Sep. 2001.

[3] Sinha, Manoj. Low-voltage sensing techniques for high-speed on-chip global interconnects and caches. Master's thesis, University of Massachusetts, Amherst, Department of Electrical and Computer Engineering, 2002.

[4] Srinivasan, Sriram. Circuit & signaling strategies for on-chip global interconnects in dsm cmos. Master's thesis, University of Massachusetts, Amherst, Department of Electrical and Computer Engineering, 2002.

[5] Kao, J., Miyazaki, M., and Chandrakasan, A. A 175-mv multipy-accumulate unit using an adaptive supply voltage and body bias architecture. *IEEE Journal of Solid-State Circuits*, 37(11), Nov. 2002.

[6] Kuroda, T., Suzuki, K., Mita, S., Fujita, T., Yamane, F., Sano, F., Chiba, A., Watanabe, Y., Matsuda, K., Maeda, T., Sakurai, T., and Furuyama, T. Variable supply-voltage scheme for low-power high-speed cmos digital design. *IEEE Journal of Solid-State Circuits*, 33(3), Mar 1999.

[7] Chandrakasan, Anantha and Brodersen, Robert. *Low-power digital CMOS Design.* Kluwer Academic Publishers, Norwell, MA, 1995.

[8] Chandrakasan, Anantha and Brodersen, Robert. *Low-power CMOS Design.* IEEE Press, Piscataway, NJ, 1998.

[9] Benini, L. and Micheli, G. De. *Dynamic Power Management, Design Techniques and Cad Tools.* Kluwer Academic Publishers, Norwell, MA, USA, 1998.

[10] Benini, L., Bruni, D., Ricco, B., Macii, A., and Macii, E. An adaptive data compression scheme for memory traffic minimization in processor-based systems. In *Proceedings, IEEE International Symposium on Circuits and Systems*, Scottsdale, AZ, USA, May 2002.

[11] Benini, L., Bogliolo, A., Paleologo, G. A., and Micheli, G. De. Policy optimization for dynamic power management. *IEEE Transactions on COMPUTER-AIDED DESIGN of Integrated Circuits and Systems*, 18(6), Jun. 1999.

[12] Benini, L. and Micheli, G. De. System-level power optimization: Techniques and tools. *ACM Transactions on Design Automation of Electronic Systems (TODAES*, Apr. 2000.

[13] Benini, L., Bogliolo, A., and Micheli, G. De. A survey of design techniques for system-level dynamic power management. *IEEE Transactions on Very Large Scale Integration Systems.*

[14] Bhardwaj, M., Min, R., and Chandrakasan, A. Power-aware systems. In *Proceedings, Thirty-Fourth Asilomar Conference on Signals, Systems, and Computers*, Pacific Grove, CA, Nov. 2000.

[15] Bhardwaj, M., Min, R., and Chandrakasan, A. Quantifying and enhancing power awareness of vlsi systems. *IEEE Transaction on Very Large Scale Integration Systems*, (10), Dec. 2002.

[16] Njolstad, Tormod and Aas, Einar. Validation of an accurate and simple delay model and its application to voltage scaling. In *Proceedings, IEEE International Symposium on Circuits and Systems*, Monterey, CA, May 1998.

[17] Pillai, Padmanabhan and Shin, Kang. Real-time dynamic voltage scaling for low-power embedded operating systems. In *Proceedings, ACM Symposium on Operating Systems Principles*, Chateau Lake Louise, Banff, Canada, Oct. 2001.

[18] Secareanu, Radu, Albonesi, David, and Friedman, Eby. A dynamic reconfigurable clock generator. In *Proceedings, IEEE International Application Specific Integrated Circuits/System-on-a-Chip Conference*, Washington, DC, Sep. 2001.

[19] Graybill, Robert and Melhem, Rami. *Power Aware Computing.* Kluwer Academic Publishers, New York, NY, 2002.

[20] Jain, P., Laffely, A., Burleson, W., Tessier, R., and Goeckel, D. Dynamically parameterized algorithms and architectures to exploit signal variations. *Journal of VLSI Signal Processing, Special Issue on Field Programmable Logic.*

[21] Short mpeg-2 description. Technical report, 2000.

[22] Mpeg-4 overview (v.21). Technical report, 2002.

[23] Jain, Prashant. Parameterized motion estimation architecture for dynamically varying power and compression requirements. Master's thesis, University of Massachusetts, Amherst, Department of Electrical and Computer Engineering, 2001.

[24] Venkatraman, Subramanian. A power-aware synthesizable core for the discrete cosine transform. Master's thesis, University of Massachusetts, Amherst, Department of Electrical and Computer Engineering, 2001.

[25] Moriyoshi, Tatsuji, Shinohara, Hiroshi, Miyazaki, Takashi, and Kuroda, Ichiro. Real-time software video codec with a fast adaptive motion vector search. *Journal of VLSI Signal Processing*, 29(3), Nov. 2001.

[26] Benini, L., Bogliolo, A., and Micheli, G. De. Dynamic power management of electronic systems. In *Proceedings, International Conference on Computer-Aided Design (ICCAD '98)*, San Jose, CA, USA, Nov. 1998.

[27] Burd, T., Pering, T., Stratakos, A., and Brodersen, R. A dynamic voltage scaled microprocessor system. *IEEE Journal of Solid-State Circuits*, 35(11), Nov. 2000.

[28] Euh, J. and Burleson, W. Exploiting content variation and perception in power-aware 3d graphics rendering. In *Proceedings, Power-Aware Computing Symposium*, Cambridge, MA, USA, Nov. 2000.

[29] Xanthopoulos, T. and Chandrakasan, A. Low-power dct core using adaptive bitwidth and arithmetic activity exploiting signal correlations and quantization. *IEEE Journal of Solid-State Circuits.*

[30] Alsolaim, Ahmad, Starzyk, Janusz, Becker, Jrgen, and Glesner, Manfred. Architecture and application of a dynamically reconfigurable hardware array for future mobile communication system. In *Proceedings, FCCM*, Napa, CA, USA, Apr. 2000.

[31] Cheng, Esther, Zhou, Feng, Yao, Bo, Cheng, Chung-Kuan, and Graham, Ronald. Balancing the interconnect topology for arrays of processors between cost and power. In *Proceedings, IEEE International Conference on Computer Design: VLSI in Computers and Processors*, Freiburg, Germany, Sep. 2002.

[32] Dally, William and Towles, Brian. Route packets, not wires: On-chip interconnection networks. In *Proceedings, 38th Design Automation Conference*, Las Vagas, NV, Jun 2001.

[33] Kumar, Shashi, Jantsch, Axel, Millberg, Mikael, berg, Johny, Soininen, Juha-Pekka, Forsell, Martti, Tiensyrj, Kari, and Hemani, Ahmed. A network on chip architecture and design methodology. In *Proceedings, IEEE Computer Society Annual Symposium on VLSI*, Pittsburgh, PA, USA, Apr. 2002.

[34] Liang, J., Swaminathan, S., and Tessier, R. asoc: A scalable, single-chip communications architecture. In *Proceedings, International Conference on Parallel Architectures and Compilation Techniques*, Philidelphia, PA, Oct. 2000.

[35] Marculescu, Radu. Networks-on-chip: The quest for on-chip fault tolerant communication. In *Proceedings, IEEE Annual Symposium on VLSI*, Tampa, FL, USA, Feb. 2003.

[36] Laffely, A., Liang, J., Jain, P., Weng, N., Burleson, W., and Tessier, R. Adaptive system on a chip (asoc) for low-power signal processing. In *Proceedings, Thirty-Fifth Asilomar Conference on Signals, Systems, and Computers*, Pacific Grove, CA, USA, Nov. 2001.

[37] Liang, J., Laffely, A., Srinivasan, S., and Tessier, R. asoc: A scalable, single-chip communications architecture. *IEEE Journal of VLSI.*

[38] Bellaouar, Abdellatif and Elmasry, Mohamed. *Low-Power VLSI Design, Circuits and Systems.* Kluwer Academic Publishers, Norwell, MA, 1995.

[39] Rabaey, Jan and Pedram, Massoud. *Low power design methodologies.* Kluwer Academic Publishers, Norwell, MA, 1996.

[40] Roy, Kaushik and Prasad, Sharat. *Low-Power CMOS VLSI Circuit Design.* John Wiley and Sons, inc., New York, NY, 2000.

[41] Augsbuger, Ann. Using dual-supply, dual-threshold and transistor sizing to reduce power in digital integrated circuits. Master's thesis, University of California at Berkeley, Department of Electrical Engineering and Computer Sciences, 2002.

[42] Bertozzi, Davide, Benini, Luca, and Ricco, Bruno. Parametric timing and power macromodels for high level simulation of low-swing interconnect. In *Proceedings, International Symposium on Low Power Electronics and Design*, Monterey, CA, USA, Aug. 2002.

[43] Sotiriadis, Paul, Konstantakopoulos, Theodoros, and Chandrakasan, Anantha. Analysis and implementation of charge recycling for deep sub-micron buses. In *Proceedings, International Symposium on Low Power Electronics and Design*, Huntington Beach, CA, USA, Aug. 2001.

[44] Stan, Mircea and Burleson, Wayne. Low-power cmos clock drivers. In *Proceedings, ACM International Workshop on Timing Issues in the Specification and Synthesis of Digital Systems*, Seattle, WA, Oct. 1995.

[45] Shahidi, Ghavam and Assaderaghi, Fari. Soi technonogy and circuits. In *Design of High Performance Microprocessor Circuits.*

[46] Burd, Thomas and Brodersen, Robert. Energy efficient cmos microprocessor design. In *Proceedings, 28th Hawaii International Conference on System Sciences*, Maui, HI, Jan. 1995.

[47] Mehra, Renu, Guerra, Lisa, and Rabaey, Jan. Low-power architecture synthesis and the impact of exploiting locality. *Journal of VLSI Signal Processing*, 1996.

[48] Wang, Alice, Chandrakasan, Anantha, and Kosonocky, S. Optimal supply and threshold scaling for subthreshold cmos circuits. In *Proceedings, IEEE Computer Society Annual Symposium on VLSI*, Pittsburgh, PA , USA, Apr. 2002.

[49] T. Hashimoto, M. Matsuo H. Nakajima Y. Kohashi K. Ishida T. Mori-iwa M. Ohashi K. Hashimoto T. Yonezawa M. Hamada T. Nakamura M. Toujima Y. Sugisawa T. Kondo H. Otsuki M. Arita H. Fujimoto H. Toida, S. Kuromaru and Ito, H. A 90mw mpeg4 video codec lsi with the capability for core profile. In *Proceedings, IEEE International Solid State Circuit Conference*, San Francisco, CA, USA, Feb. 2000.

[50] Nishikawa, T., Takahashi, M., Hamada, M., Takayanagi, T., Arakida, H., Machida, N., Yamamoto, H., Fujiyoshi, T., Matsumoto, Y., Yamagishi, O., Samata, T., Asano, A., Terazawa, T., Ohmori, K., Shirakura, J., Watanabe, Y., Nakamura, H., Minami, S., Kuroda, T., , and Furuyama, T. A 60mhz 240mw mpeg-4 video-phone lsi with 16mb embedded dram. In *Proceedings, IEEE International Solid State Circuit Conference*, San Francisco, CA, USA, Feb 2000.

[51] Zhao, J., Chandramouli, R., Vijaykrishnan, N., Irwin, M.J., Kang, B., and Somasundaram, S. Influence of mpeg-4 parameters on system energy. In *Proceedings, IEEE International Application Specific Integrated Circuits/System-on-a-Chip Conference*, Rochester, NY, Sep. 2002.

[52] Stan, Miccea and Burleson, Wayne. Bus-invert coding for low-power i/o. Mar. 1995.

[53] Unsal, Osman S., Koren, Israel, Krishna, C. Mani, and Moritz, Csaba Andras. Cool-fetch: A compiler-based ipc estimation based framework for energy-efficiency. *ACM Computer Architecture Letters*, 1.

[54] Sinha, Amit and Chandrakasan, Anantha. Energy aware software. In *Proceedings, International Conference on VLSI Design*, Calcutta, India, Jan. 2000.

[55] Sinha, Amit, Wang, Alice, and Chandrakasan, Anantha P. Algorithmic transforms for efficient energy scalable computation. In *Proceedings,*.

[56] Sinha, Amit and Chandrakasan, Anantha P. Dynamic voltage scheduling using adaptive filtering of workload traces.

[57] Sinha, Amit, Wang, Alice, and Chandrakasan, Anantha. Energy scalable system design. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 10, Apr. 2002.

[58] Hsu, C. and Kremer, U. Compiler-directed dynamic voltage scaling for memory-bound applications. *Rutgers University Technical Report DCS-TR498*, Aug. 2002.

[59] Simunic, Tajana, Benini, Luca, Acquaviva, Andrea, Glynn, Peter, and Micheli, Giovanni De. Dynamic voltage scaling and power management for portable systems. In *Proceedings, 38th Design Automation Conference*, Las Vagas, NV, Jun. 2001.

[60] Son, Donghwan, Yu, Chansu, and Kim, Heung-Nam. Dynamic voltage scaling on mpeg decoding. In *Proceedings, International Conference on Parallel and Distributed Systems*, KyongJu City, Korea, Jun. 2001.

[61] Nawab, S. Hamid, Oppenheim, Alan, Chandrakasan, Anantha, Winograd, Joseph, and Ludwig, Jeffrey. Approximate signal processing. *Journal of VLSI Signal Processing*, 15(1-2), Jan. 1997.

[62] Rabaey, J. Reconfigurable processing: The solution to low-power programmable dsp. In *Proceedings, International Conference on Acoustics, Speech and Signal Processing*, Munich, Germany, Apr. 1997.

[63] Zhang, Hui, Prabhu, Vandana, George, Varghese, Wan, Marlene, Benes, Martin, Abnous, Arthur, and Rabaey, Jan. A 1v heterogeneous reconfigurable dsp ic for baseband wireless digital signal processing. *IEEE Journal on Solid State Circuits*, 35(11), Nov. 2000.

[64] Ashok, R., Chheda, S., and Moritz, C. A. Cool-mem: Combining statically speculative memory accessing with selective address translation for energy efficiency. In *Proceedings, ASPLOS*, San Jose, CA, USA, Oct. 2002.

[65] Unsal, Osman S., Koren, Israel, Krishna, C. Mani, and Moritz, Csaba Andras. The minimax cache: An energy-efficient framework for media processors. In *Proceedings, International Symposium on High-Performance Computer Architecture*, Boston, MA, USA, Feb. 2002.

[66] Xanthopoulos, T. and Chandrakasan, A. A low-power idct macrocell for mpeg-2 mpml exploiting data distribution properties for minimal activity. *IEEE Journal of Solid-State Circuits*, 34, May 1999.

[67] Xanthopoulos, T. and Chandrakasan, A. A low-power dct core using adaptive bitwidth and arithmetic activity exploiting signal correlations and quantization. In *Proceedings, Symposium on VLSI Circuits*, Kyoto, Japan, Jun 1999.

[68] Cao, Y., Sato, T., Sylvester, D., Orchansky, M., and Hu, C. New paradigm of predictive mosfet and interconnect modeling for early circuit design. In *Proceedings, IEEE Custom Integrated Circuits Conference*, Orlando, FL, Jun. 2000.

[69] Rabaey, Jan. *Digital Integrated Circuits, A Design Perspective.* Prentice Hall Inc., Upper Saddle River, NJ, 1996.

[70] Gronowski, Paul, Bowhill, William, Preston, Ronald, Growan, Michael, and Allmon, Randy. High-performance microprocessor design. *IEEE Journal of Solid-State Circuits*, 33(5), May 1998.

[71] Altera Corporation. *Apex20KE Device DataSheet*, 2002.

[72] Xilinx Corporation. *Virtex II Device DataSheet*, 2002.

[73] Kaviani, A. and Brown, S. Technology mapping issues for an fpga with lookup tables and pla-like blocks. In *ACM/SIGDA 8th International Symposium on Field-Programmable Gate Arrays*, pages 60–66, Feb. 2000.

[74] De, Vivek, Ye, Yibin, Keshavarzi, Ali, Narendra, Siva, Kao, James, Somasekhar, Dinesh, Nair, Raj, and Borkar, Shekhar. Techniques for leakage power reduction. In *Design of High Performance Microprocessor Circuits.*

[75] Maheshwari, Atul, Srinivasaraghavan, Srividya, and Burleson, Wayne. Quantifying the impact of current-sensing on interconnect delays trends. In *Proceedings, IEEE International Application Specific Integrated Circuits/System-on-a-Chip Conference*, Rochester, NY, USA, Sep. 2002.

[76] Favrat, P., Deval, P., and Declercq, M. A high-efficiency cmos voltage doubler. *IEEE Journal of Solid-State Circuits*, 33(3), Mar. 1998.

[77] Gutnik, V. and Chandrakasan, A. Embedded power supply for low-power dsp. *IEEE Transactions on Very Large Scale Integration*, 5(4), Dec. 1997.

[78] Ichiba, F., Suzuki, K., Mita, S., Kuroda, T., and Furuyama, T. Variable supply-voltage scheme with 95%-efficiency dc-dc converter for mpeg-4 codec. In *Proceedings, International Symposium on Low Power Electronics and Design*, Aug. 1999.

[79] Ker, M., Chang, C., and Jiang, H. Design of negative charge pump circuit with polysilicon diodes in a 0.25-um cmos process. In *Proceedings, ASIA-Pacific Aplication Specific Integrated Circuits Conference.*

[80] Pelliconi, Roberto, Iezzi, David, Baroni, Andrea, Pasotti, Marco, and Rolandi, Pier Luigi. Power efficient charge pump in deep submicron standard cmos technology. In *Proceedings, 27th Eropean Solid State Circits Conference*, Villach, Austria, Sep. 2001.

[81] *Arm: The Archiecture for the Digital World.*

[82] Sinha, Amit and Chandrakasan, Anantha P. Jouletrack-a web based tool for software energy profiling. In *Proceedings,*, Las Vegas, NV, USA, Jun. 2001.

[83] Short mpeg-1 description. Technical report, 1996.

[84] Mpeg-7 overview (v.8). Technical report, 2002.

[85] Mpeg-21 overview (v.5). Technical report, 2002.

[86] *Understanding Industrial Designed Experiments.* Air Academy Press, 1994.

[87] Benini, L., Bruni, D., Macii, A., and Macii, E. Hardware implementation of data compression algorithms for memory energy optimization. In *Proceedings, IEEE Computer Society Annual Symposium on VLSI*, Tampa, FL, USA, Feb. 2003.

[88] Chan, F. and Haccoun, D. Adaptive viterbi decoding of convolutional codes over memoryless channels. *IEEE Transactions on Communications*, 45, Nov. 1997.

[89] Swaminathan, S., Tessier, R., Goeckel, D., and Burleson, W. A dynamically reconfigurable adaptive viterbi decoder. In *Proceedings, International ACM/SIGDA Symposium on Field Programmable Gate Arrays*, Monterey, CA, USA, Feb. 2002.

[90] Simmons, S. Breadth-first trellis decoding with adaptive effort. *IEEE Transactions on Communications*, 38, 2000.

[91] Kuhn, Peter. *Algorithms, Complexity Analysis and VLSI Architectures for MPEG-4 Motion Estimation.* Kluwer Academic Publishers, Norwell, MA, 1999.

[92] Jung, B. and Burleson, W. Vlsi algorithm, architecture and implementation for high-speed lempel-ziv data compression. *IEEE Transactions on VLSI Systems*, Sep. 1998.

[93] Bhaskaran, V. and Konstantinides, K. *Image and Video Compression Standards, Algorithms and Architectures, Second Edition.* Kluwer Academic Publishers, Norwell, MA, 1997.

[94] Jung, B. *VLSI Arrays for Source Coding in Wireless Local Area Networks.* PhD thesis, University of Massachusetts Amherst, Feb. 1997.

[95] Viterbi, A. Error bound for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE Transactions on Information Theory*, 13, Apr. 1967.

[96] Omura, J. On the viterbi decoding algorithm. *IEEE Transactions on Information Theory*, 15, Apr. 1969.

[97] Corporation, Xilinx. Virtex ii data sheet. *http://www.xilinx.com*, 2001.

[98] Ackland, B., Anesko, A., Brinthaupt, D., Daubert, S., Kalavade, A., Knobloch, J., Micca, E., Moturi, M., Nicol, C., ONeill, J, Othmer, J., Sackinger, E., Singh, K., Sweet, J., Terman, C., and Williams, J. A single-chip,

1.6-billion, 16-b mac/s multiprocessor dsp. *Journal of Solid-State Circuits*, 35(3), Mar. 2000.

[99] *Calisto Product Brief.*

[100] Adams, Lyle. Overview of the coreframe architecture. Technical report, PalmChip Corporation, 2000.

[101] Brynjolfson, Ian and Zilic, Zeljko. Mcsoc: A platform for clock managed systems on a chip. In *Proceedings, IEEE International ASIC-SOC Conference*, Washington, DC, USA, Sep. 2001.

[102] Leijten, Jeroen, van Meerbergen, Jef, Timmer, Adwin, and Jess, Jochen. Prophid: A platform-based design method. *Design Automation for Embedded Systems*, 6(1), Sep. 2000.

[103] Leijten, Jeroen, van Meerbergen, Jef, Timmer, Adwin, and Jess, Jochen. Stream communication between real-time tasks in a high-performance multiprocessor. In *Proceedings, Design Automation and Test in Europe Conference*, Paris, France, Mar. 1998.

[104] Abu-Ghazaleh, Nael B. and Wilsey, Philip A. The shared control parallel architecture model. *Journal of Parallel and Distributed Computing*, 61(6), Jun. 2001.

[105] Aldworth, P. J. System-on-a-chip bus architecture for embedded applications. In *Proceedings, IEEE International Conference on Computer Design*, Austin, TX, USA, Oct. 2000.

[106] Bergamaschi, Reinaldo and Lee, William. Desiging systems-on-chip using cores. In *Proceedings, 37th Design Automation Conference*, Los Angeles, CA, USA, Jun. 2000.

[107] Givargis, Tony, Vahid, Frank, and Henkel, Jrg. A hybrid approach for core-based system-level power modeling. In *Proceedings, Asia South Pacific Design Automation Conference*, Yokohama, Japan, Jan. 2000.

[108] *IDT Peripheral Bus: Intermodule Connection Technology Enables Broad Range of System-Level Integration.*

[109] Theis, T. The future of interconnection technology. *IBM Journal of Research and Development*, May 2000.

[110] Iyer, Anoop and Marculescu, Diana. Microarchitecture-level power management. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 10, Jun. 2002.

[111] Iyer, Anoop and Marculescu, Diana. Power and performance evaluation of globally asynchronous locally synchronous processors. In *Proceedings, International Symposium on Computer Architecture*, Anchorage, AK, USA, May 2002.

[112] Iyer, Anoop and Marculescu, Diana. Power efficiency of voltage scaling in multiple clock, multiple voltage cores. In *Proceedings, IEEE/ACM International Conference on Computer Aided Design*, San Jose, CA, USA, Nov. 2002.

[113] Betz, V. and Rose, J. Cluster-based logic block for fpgas: Area-efficency vs. input sharing and size. In *Proceedings, International Custom Integrated Circuits Conference*, Santa Clara, CA, USA, May 1997.

[114] David, Raphael, Chillet, Daniel, Pillement, Sebastien, and Sentieys, Olivier. Dart: A dynamically reconfigurable architecture dealing with future mobile telecommunications constraints. In *Proceedings, International Parallel and Distributed Processing Symposium*, Fort Lauderdale, Fl, USA, Apr. 2002.

[115] Cherepacha, Don and Lewis, David. A datapath oriented architecture for fpgas. In *Proceedings, ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, Monterey, CA, USA, Feb. 1994.

[116] Marshall, Allan, Stansfield, Tony, Kostarnov, Igor, Vuillemin, Jean, and Hutchings, Brad. A reconfigurable arithmetic array for multimedia applications. In *Proceedings, ACM/SIGDA Seventh International Symposium on Field Programmable Gate Arrays*, Monterey, Ca, USA, Feb. 1999.

[117] Mirsky, Ethan and DeHon, Andr. Matrix : A reconfigurable computing architecture with configurable instruction distribution and deployable resources. In *Proceedings, IEEE Symposium on FPGAs for Custom Computing Machines*, Napa, CA, USA, Apr 1996.

[118] Benoit, Pascal, Sassatelli, Gilles, Robert, Michel, Torres, Lionel, Cambon, Gaston, and Gilles, Thierry. The systolic ring : A scalable dynamically reconfigurable core for embedded systems. In *Proceedings, Sophia Antipolis MicroElectronics Forum*, Sophia Antipolis, France, Oct. 2002.

[119] Sassatelli, Gilles, Torres, Lionel, Benoit, Pascal, Cambon, Gaston, Robert, Michel, and Galy, Jrme. Dynamically reconfigurable architectures for digital signal processing applications. In *Proceedings, IFIP International Conference on Very Large Scale Integration The Global System on Chip Design and CAD Conference1*, Montpellier, France, Dec. 2001.

[120] Sassatelli, Gilles, Torres, Lionel, Benoit, Pascal, Gil, Thierry, Diou, Camille, Cambon, Gaston, and Galy, Jrme. Highly scalable dynamically reconfigurable systolicring-architecture for dsp applications. In *Proceedings, IEEE Design, Automation and Test in Europe*, Paris, France, Mar. 2002.

[121] Bittner, Ray, Athanas, Peter, and Musgrove, Mark. Colt: An experiment in wormhole run-time reconfiguration. In *Proceedings, SPIE Photonics East*, Boston MA, USA, Nov. 1996.

[122] Chen, D. and Rabaey, J. Paddi: Programmable arithmetic devices for digital signal processing. *IEEE VLSI Signal Processing*, IV, 1990.

[123] Yeung, A. K. W. and Rabaey, J.M. A reconfigurable data-driven multiprocessor architecture for rapid prototyping of high throughput dsp algorithms. In *Proceedings, HICSS-26*, Kauai, Hawaii, Jan. 1993.

[124] *The D-Fabrix Array.*

[125] Ebeling, Carl, Cronquist, Darren C., and Franklin, Paul. Rapid - reconfigurable pipelined datapath. In *Proceedings, Sixth International Workshop on Field Programmable Logic and Applications*, Darmstadt, Germany, Sep. 1996.

[126] Forsell, M. High-performance computing solution for network-on-chips. *IEEE Micro*, 22(5), Sep. - Oct. 2002.

[127] Goldstein, Steh Copen, Schmit, Herman, Moe, Matthew, Budiu, Mihai, Cadambi, Srihari, Taylor, R. Reed, and Laufer, Ronald. Piperench : A coprocessor for streaming multimedia acceleration. In *Proceedings, 26th International Symposium on Computer Architecture*, Atlanta, GA, USA, May 1999.

[128] Hartenstein, R., Hoffmann, Th., and Nageldinger, U. Design-space exploration of low power coarse grained reconfigurable datapath array architectures. In *Proceedings, International Workshop - Power And Timing Modeling, Optimization and Simulation*, Germany, Sep. 2000.

[129] Hartenstein, R., Hoffmann, Th., and Nageldinger, U. Kressarray xplorer : A new cad environment to optimize reconfigurable datapath array architectures. In *Proceedings, 5th Asia and South Pacific Design Automation Conference*, Yokohama, Japan, Jan. 2000.

[130] Hartenstein, R., Hoffmann, Th., and Nageldinger, U. Generation of design suggestions for coarse-grain reconfigurable architectures. In *Proceedings, FPL*, 2000.

[131] Hauser, John R. and Wawrzynek, John. Garp : A mips processor with a reconfigurable coprocessor. In *Proceedings, The 5th Annual IEEE Symposium on Field-Programmable Custom Computing Machines*, Napa, CA, USA, Apr. 1997.

[132] Heysters, Paul, Smit, Japp, Smit, Gerard, and Havinga, Paul. Mapping of dsp algorithms on field programmable function arrays. In *Proceedings, Tenth International Workshop on Field Programmable Logic and Applications*, Villach, Austria, Aug. 2000.

[133] Heysters, Paul M., Smit, Japp, Smit, Gerard J.M., and Havinga, Paul J.M. Exploiting energy-efficient reconfigurable architectures for dsp algorithms. In *Proceedings, Progress 2000 Workshop*, Oct 2000.

[134] Miyamori, T. and Olukotun, K. Remarc: Reconfigurable multimedia array coprocessor. In *Proceedings, ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, Monterey, CA, USA, Feb. 1998.

[135] Singh, Hartej, Lu, Guangming, Lee, Ming-Hau, Kurdahi, Fadi, Bagherzadeh, Nader, Filho, Eliseu, and Mestre, Rafael. Morphosys : Case study of a reconfigurable computing systems targeting multimedia applications. In *Proceedings, ASP-DAC*, Yokohama, Japan, Jan. 2000.

[136] Waingold, Elliot, Taylor, Michael, Sarkar, Vivek, Lee, Walter, Lee, Victor, Kim, Jang, Frank, Matthew, Finch, Peter, Devabhaktuni, Srikrishna, Barua, Rajeev, Babb, Jonathon, Amarasinghe, Saman, and Agarwal, Anant. Baring it all to software: Raw machines. *IEEE Computer*, 30(9), Sep. 1997.

[137] Lee, Walter, Barua, Rajeev, Frank, Matthew, Srikrishna, Devabhaktuni, Babb, Jonathan, Sarkar, Vivek, and Amarasinghe, Saman. Space-time scheduling of instruction-level parallelism on a raw machine. In *Proceedings, the Eighth International Conference on Architectural Support for Programming Languages and Operating Systems*, San Jose, CA, Oct. 1998.

[138] Moritz, Csaba Andras, Yeung, Donald, and Agarwal, Anant. Simplefit: A framework for analyzing design trade-offs in raw architectures. *IEEE Transactions on Parallel and Distributed Systems*, 12(6), Jun. 2001.

[139] Taylor, Michael Bedford, Lee, Walter, Amarasinghe, Saman, and Agarwal, Anant. Scalar operand networks: On-chip interconnect for ilp in partitioned architectures. In *Proceedings, The International Symposium on High Performance Computer Architecture*, Anaheim, CA, USA, Feb 2003.

[140] Taylor, Michael Bedford, Kim, Jason, Miller, Jason, Wentzlaff, David, Ghodrat, Fae, Greenwald, Ben, Hoffmann, Henry, Johnson, Paul, Lee, Walter, Saraf, Arvind, Shnidman, Nathan, Strumpen, Volker, Amarasinghe, Saman, and Agarwal, Anant. A 16-issue multiple-program-counter microprocessor with point-to-point scalar operand network. In *Proceedings, The IEEE International Solid-State Circuits Conference*, San Francisco, CA, USA, Feb. 2003.

[141] Taylor, Michael Bedford, Kim, Jason, Miller, Jason, Wentzlaff, David, Ghodrat, Fae, Greenwald, Ben, Hoffmann, Henry, Lee, Jae-Wook, Johnson, Paul, Lee, Walter, Ma, Albert, Saraf, Arvind, Seneski, Mark, Shnidman, Nathan, Strumpen, Volker, Frank, Matt, Amarasinghe, Saman, and Agarwal, Anant. The raw microprocessor: A computational fabric for software circuits and general purpose programs. *IEEE Micro*, Mar/Apr 2002.

[142] *Reconfigurable Communications Processor.*

[143] *FIPSOC Mixed Signal System-on-Chip.*

[144] Benini, Luca and Micheli, Giovanni De. Networks on chips: a new soc paradigm. *Computer*, 35(1), Jan. 2002.

[145] Benini, Luca and Micheli, Giovanni De. Networks on chip: a new paradigm for systems on chip design. In *Proceedings, Design, Automation and Test in Europe Conference and Exhibition*, Paris, France, Mar. 2002.

[146] Benini, Luca and Micheli, Giovanni De. Powering networks on chips. In *Proceedings, The 14th International Symposium on System Synthesis*, Montreal, Que., Canada, Sep. 2001.

[147] Darringer, J., Bergamaschi, R., Bhattacharya, S., Brand, D., Herkersdorf, A., Morrell, J., Nair, I., Sagmeister, P., and Shin, Y. Early analysis tools for system-on-a-chip design. *IBM Journal of Research and Development*, 46(6), Nov.

[148] Hu, Jingcao, Deng, Yangdong, and Marculescu, Radu. System-level point-to-point communication synthesis using floorplanning information. In *Proceedings, ASP-DAC*, Jan 2002.

[149] Lundheim, Lars, Njolstad, Tormod, and Tjore, Odd. On the efficiency of dynamic voltage scaling in a digital front end for a burst-based radio receiver. In *Proceedings, International Symposium on Telecommunications Mobile Summit*, Barcelona, Spain, Sep. 2001.

[150] Njolstad, Tormod, Tjore, Odd, Svarstad, Kjetil, Lundheim, Lars, Vedal, Tor O., Typpo, Jukka, Ramstad, Tor, Wanhammar, Lars, Aas, Einar J., and Danielsen, Helge. A socket interface for gals using locally dynamic voltage scaling for rate-adaptive energy savings. In *Proceedings, Fourteeth Annual IEEE International ASIC/SOC Conference*, Arlington, VA, Sep. 2001.

[151] Yun, Kenneth and Dooply, Ayoob. Pausible clocking-based heterogeneous systems. *IEEE Transactions on Very Large Scale Integration Systems*, 7(4), Dec. 1999.

[152] Ryu, Kyeong and III, Vincent Mooney. Automated bus generation for multiprocessor soc design. Technical report, CREST Technical Report CREST-TR-02-005, 2003.

[153] Shalan, M. and Mooney, V. Hardware support for real-time embedded multiprocessor system-on-a-chip memory management. In *Proceedings, Tenth International Symposium on Hardware/Software Codesign*, Estes Park, CO, USA, May 2002.

[154] *The CoreConnect Bus Architecture.*

[155] *The WISHBONE Service Center.*

[156] *Virtual Socket Interface Alliance.*

[157] Flynn, David. Amba: Enabling reusable on-chip design. *IEEE Micro*, Jul. 1997.

[158] Budiu, Mihai and Goldstein, Seth Copen. Fast compilation for pipelined reconfigurable fabrics. In *Proceedings, the 1999 ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, Monterey, CA, USA, Feb. 1999.

[159] Amerijckx, C. and Legat, J. A low-power multiprocessor architecture for embedded reconfigurable systems. In *Proceedings, The International Workshop on Power and Timing Modeling, Optimization and Simulation*, Denmark, Oct. 1998.

[160] Myers, Chris. *Asynchronous Circuit Design.* John Wiley and Sons INC., New York, NY, USA, 2001.

[161] Muttersbach, Jens, Villiger, Thomas, Kaeslin, Hubert, Felber, Norber, and Fichtner, Wolfgang. Globally-asynchronous locally-synchronous architectures to simplify the design of on-chip systems. In *Proceedings, IEEE International ASIC/SOC Conference*, Washington, DC, Sept. 1999.

[162] Muttersbach, Jens, Villiger, Thomas, and Fichtner, Wolfgang. Practical design of globally-asynchronous locally-synchronous systems. In *Proceedings, 6th International Symposium on Advanced Research in Asynchronous Circuits and Systems*, Eilat, Israel, Apr. 2000.

[163] Sjogren, Allen and Myers, Chris. Interfacing synchronous and asyncronous modules within a high-speed pipeline. *IEEE Transactions on Very Large Scale Integration Systems*, 8(5), Oct 2000.

[164] *IC Prototype Fabrication and Low Volume Production.*

[165] Boning, Duane and Nassif, Sani. Models of process variations in device and interconnect. In *Design of High Performance Microprocessor Circuits.*

[166] Tam, Simon, Rusu, Stefan, Desai, Utpal, Kim, Robert, Zhang, Ji, and Young, Ian. Clock generation and distribution for the first ia-65 microprocessor. *IEEE Journal of Solid-State Circuits*, 35(11), Nov. 2000.

[167] Blaauw, David, Panda, Rajendran, and Chaudhry, Rajat. Design and analysis of power distribution networks. In *Design of High Performance Microprocessor Circuits.*

[168] Secareanu, Radu, Friedman, Eby, Becerra, Juan, and Warner, Scott. A universal cmos voltage interface circuit. In *Proceedings, EEE International Symposium on Circuits and Systems*, Orlando, FL, May. 1999.

[169] Metcalf, Christopher. The numesh simulator. *MIT Lab for Computer Science, NuMesh Memo 5, http://www.cag.lcs.mit.edu/numesh/papers/memos*, Jan. 1992.

APPENDIX A

ACRONYMS

- AGU: Address Generator Unit
- ALU: Arithmatic Logic Unit
- aSoC: Adaptive System-on-a-Chip
- AVA: Adaptive Viterbi Algorithm
- BER: Bit Error Rate
- BPTM: Berkeley Predictive Technology Models
- CCIM: Communication Controller and Instruction Memory
- CDM: Communication Data Memory
- CI: Communication Interface
- CPU: Central Processing Unit
- DCT: Discrete Cosine Transform
- DIBL: Drain Induced Barrier Lowering
- DPM: Dynamic Power Management
- DSP: Digital Signal Processors or Processing
- DVS: Dynamic Voltage Scaling

- FPGA: Field Programmable Gate Array
- FSA: Full Search Algorithm
- GALS: Globally Asynchronous Locally Synchronous
- I/O: Input and Output
- IP: Intellectual Property
- jpc: Jump PC Value
- LUT: look-up-tables LUT
- LZ: Lempel-Ziv
- ME: Motion Estimation
- MPEG: Motion Picture Expert Group
- NoC Network-on-Chip
- PC: Program Counter
- PLA: Programmable Logic Arrays
- RCC: Row-Column Classification
- RISK: Reduced Instruction Set Computer
- SAD: Sum of Absolute Differences
- sj: Scheduled Jump
- SNR: Signal to Noise Ratio
- SoC: System-on-a-Chip

- SOI: Silicon on Insulator
- TSS: 3 Step Search
- v_{dd}: Supply Voltage
- VA: Viterbi Algorithm
- VLSI: Very Large Scale Integration